4.4. Functional Analysis *(Satisfies EIA/IS 731 FA 1.2 and iCMM PA 4)*

Functional Analysis details the use of functional flow diagramming as a representative structured analysis process that is the preferred approach of the Federal Aviation Administration (FAA). In addition, this section covers several alternative approaches, as FAA system engineers come in contact with organizations that apply techniques other than functional flow diagramming. Therefore, it is necessary that the engineers be able to communicate with members of those organizations and integrate their results with the work performed by other organizations. The following paragraphs detail functional flow diagramming; alternative approaches appropriate for systems and hardware; alternative models for problems to be solved with computer software; and references that cover these techniques in more depth.

4.4.1. Introduction to Functional Analysis

The process of analyzing functions provides System Engineering (SE) with a functional system description that becomes a framework for developing requirements and physical architectures. Utilizing the Functional Analysis process significantly improves synthesis of design, innovation, requirements development, and integration. The Functional Analysis process provides two key benefits to SE: (1) it discourages single-point solutions, and (2) it describes the behaviors that lead to requirements and physical architectures. The essential elements of Functional Analysis are illustrated in Figure 4.4-1, which lists the key inputs necessary to initiate the task, providers, process tasks, outputs required, and customers of process outputs. The beginning and ending boundary tasks, as well as the intermediate tasks, are described in later paragraphs.

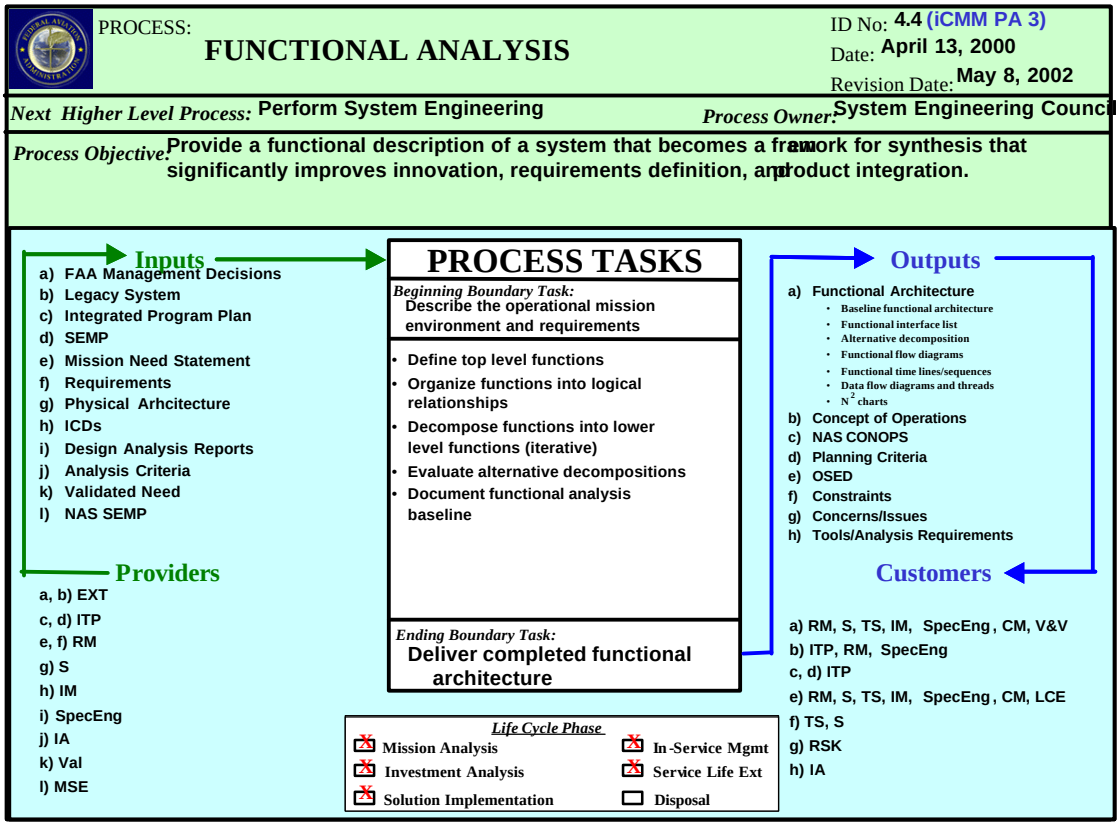


Figure 4.4-1. Functional Analysis Process-Based Management Chart

Systems may be described using two different facets. First, a system may be described as a physical architecture with elements that interact with themselves and the system environment in

accordance with a predefined process to achieve the system mission. At the same time, a system may be described by the functions that it performs. A system is intended to satisfy predefined functions, with the highest-level function defined as the stakeholder need (also the ultimate system requirements). A *function* is a characteristic action or activity that has to be performed in order to achieve a desired system objective (or stakeholder need). A *function name* is stated in the form of an action verb followed by a noun or noun phrase; it is an action that describes the desired system behavior. Examples of common functions include "read book," "eat food," and "go to store." A function is accomplished by one or more system elements composed of equipment (hardware, software, and firmware), people, and procedures. The function occurs within the system environment and is performed to achieve system operations. In Functional Analysis, because a function may be accomplished by more than one system element, functions are unable to be allocated. Rather, functions are used to develop requirements, which are then allocated to solutions in the form of a physical architecture.

When unprecedented systems or systems are being developed that radically differ from those currently in use, the approach named "form follows function" is applied. The first function to identify stems from the need, which is then decomposed into lower levels of needed functionality. The functional description is translated into the physical by assigning functionality to requirements and requirements into a Physical Architecture. While function names may be allocated to specific Physical Architecture entities directly, it is often the case that some combination of two or more architectural entities accomplishes one function. The FAA preference is to translate functions into primitive performance requirements and then allocate these performance requirements to physical architecture entities.

4.4.1.1. Functional Analysis Objectives

The Functional Analysis process helps to ensure that:

- All facets of a system's lifecycle, as illustrated in Figure 4.4-1, are covered from development to production, operation, and support
- All functional elements of the system are described, recognized, and defined
- All system concepts and requirements for specific system functions are related
- Requirements definition is improved
- Product integration is improved
- New and innovative designs and solutions are incorporated

4.4.1.2. Process Overview

Functional Analysis examines a system's functions and subfunctions that are necessary to accomplish the system's operation or mission. It describes *what* the system does, not *how* it does it. Functional Analysis is conducted at the level needed to support later synthesis efforts, with all operational modes and environments included. Each function required to meet the operational needs of a system is identified and defined; once defined, the functions are then used to define the system requirements, and a functional architecture is developed based on the identified requirements. The process is then taken to a greater level of detail as the identified functions are further decomposed into subfunctions, and the requirements and physical architecture associated with those functions are each decomposed as well. This process is iterated until the system is completely decomposed into basic subfunctions, and each subfunction at the lowest level is completely, simply, and uniquely defined by its requirements. In this process, the interfaces between each of the functions and subfunctions are fully defined, as are the interfaces with the environment and external systems. The functions and

subfunctions are arrayed in a Functional Architecture to show their relationships and interfaces (internal and external). Figure 4.4-2 illustrates the Functional Analysis process flow.

Functions shall be:

- Arranged in their logical sequence
- Well defined in their inputs, outputs, and functional interfaces (internal and external)
- Traceable from beginning to end conditions
- Analyzed, determined, and defined for time-critical requirements
- Successively established from the highest to lowest level for each function and interface
- Defined in terms of what needs to be accomplished in verb–noun combinations without describing *how* it is to be accomplished
- Traceable downward through successive functional decompositions

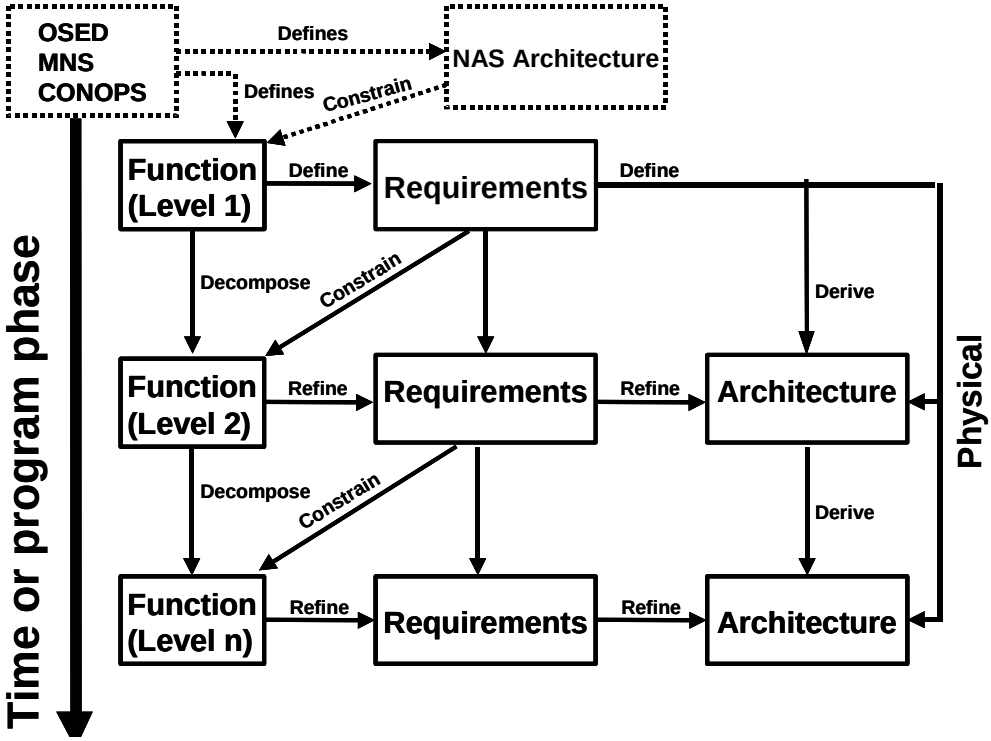


Figure 4.4-2. Functional Analysis Process Flow and Interface with Physical Architecture and Requirements

It is recommended that the Functional Analysis process be conducted in conjunction with Requirements Management (Section 4.3), Synthesis (Section 4.5), and Trade Studies (Section 4.6) (Figure 4.4-3) to:

- Define successively lower-level functions required to satisfy higher-level requirements and to define increasingly detailed sets of Functional Architectures

- Define mission- and environment-driven performance requirements and determine that higher-level requirements are satisfied
- Flow down performance requirements and design constraints
- Refine the definition of product and process solutions

4.4.2. Inputs to Functional Analysis

The more that is known about a system, the more complete the Functional Architecture. At the highest level of Functional Analysis for the FAA (the National Airspace System (NAS)), only the Mission Need Statement (MNS) may be available as input. The needs reflected in the MNS are translated into a Concept of Operations (CONOPS). A CONOPS is a high-level form of Functional Analysis that is solely derived from the user's perspective. It is recommended that the CONOPS serve as a baseline for the more detailed Functional Analyses to follow. (Paragraph 4.4.4.2 provides more information on CONOPS.) As iterations progress, it is recommended that higher-level Physical Architectures and Requirements be considered as they become available. If the output of the Requirements Management (Section 4.3) task is incomplete, the Functional Analysis task reveals missing Requirements and helps to refine or clarify others. Figure 4.4.3 depicts Functional Analysis's process flow, while Figures 4.4-4 and 4.4-5 illustrate several representative inputs and outputs to/from Functional Analysis.

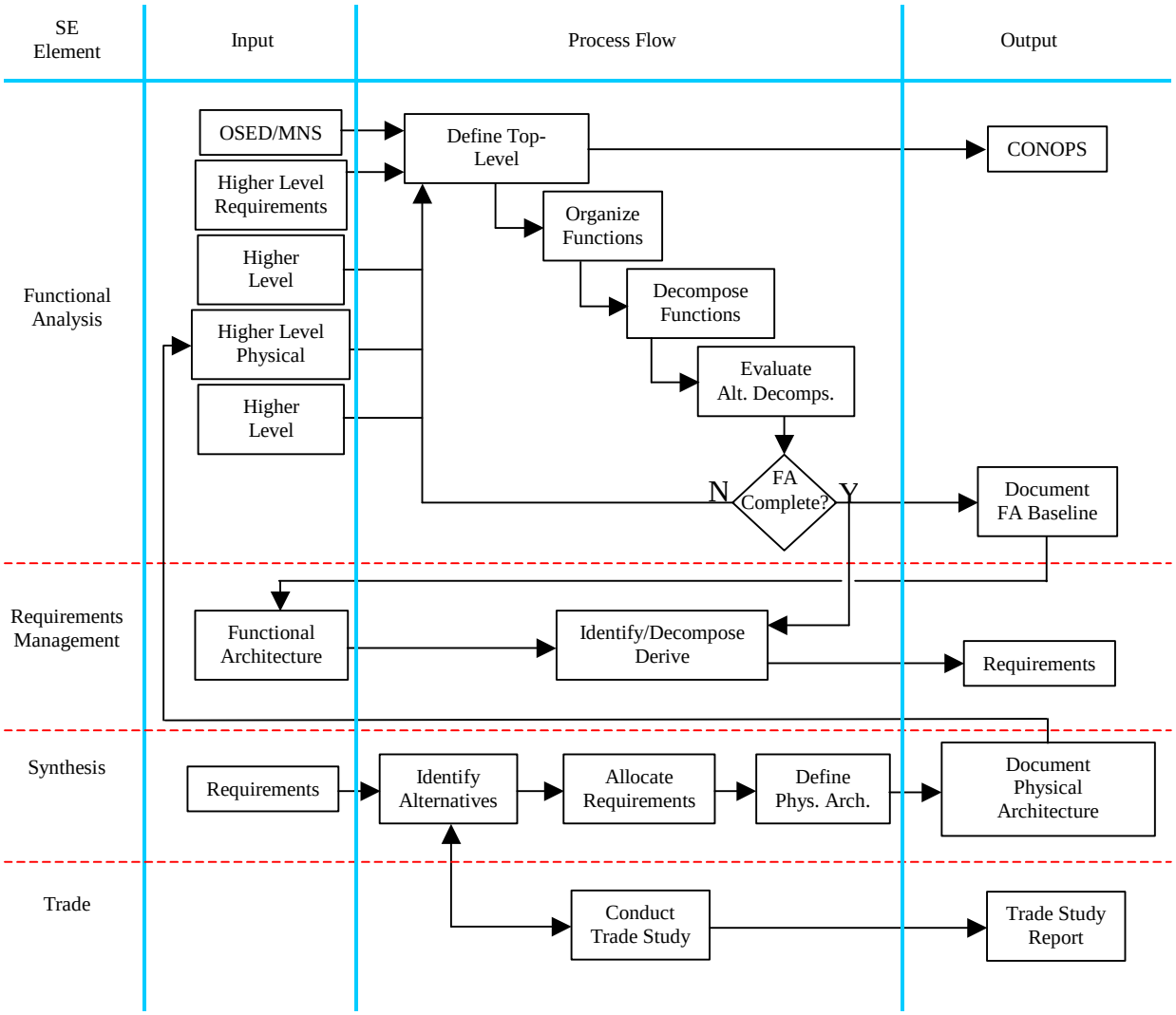


Figure 4.4-3. Functional Analysis Process Flow

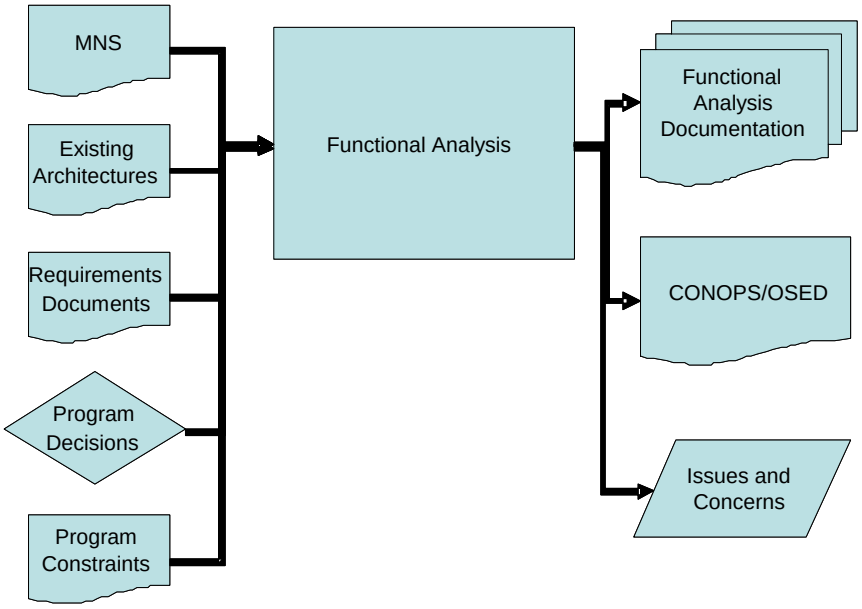


Figure 4.4-4. Several Representative Inputs to Functional Analysis

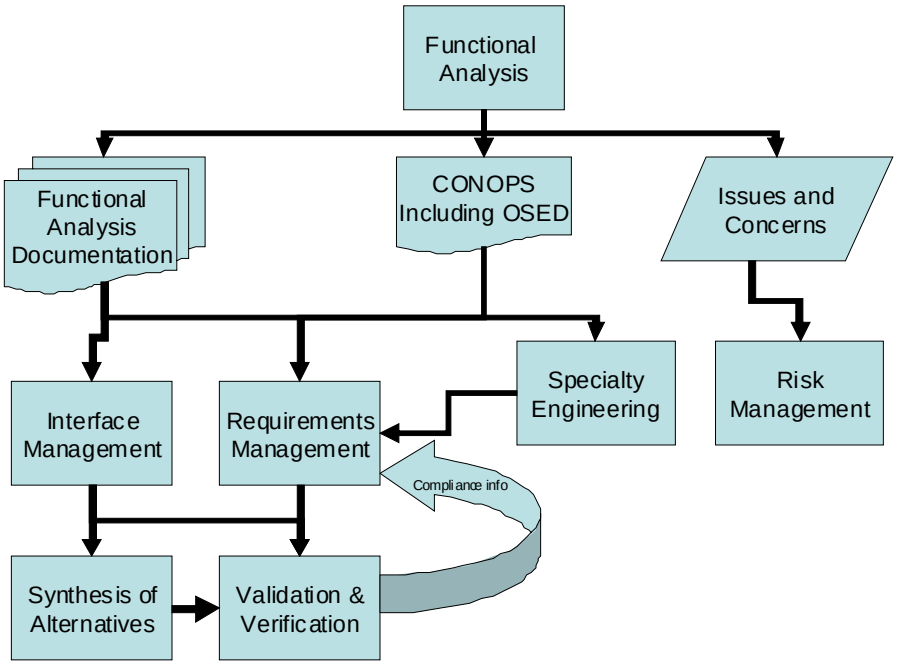


Figure 4.4-5. Several Representative Outputs of Functional Analysis

The following is a more comprehensive list of the inputs to Functional Analysis:

- FAA Management Decisions
- Information on Legacy Systems
- Planning documents, such as the Integrated Program Plan

- NAS Level (and program, if available) System Engineering Management Plan
- Defined NAS capability shortfalls and/or needs in the MNS (including validated needs statement)
- Requirements, such as any existing specifications and standards requirements, including requirements documents (reference documents)
- Program decisions (such as Constraints relating to existing hardware and software)
- Existing Physical Architectures
- Higher-level Functional Architectures
- Information on interfaces, including Interface Control Documents
- Design Analysis Reports
- Analysis Criteria

4.4.3. Functional Analysis Process Tasks

The Functional Analysis process is summarized in Figure 4.4-1. The five major process tasks listed in Figure 4.4-1 are described in the remainder of this section.

4.4.3.1. Task 1: Define Top-Level Functions (From Inputs)

The first task in defining the system from a functional standpoint is to review the MNS, existing Operational Services and Environmental Descriptions (OSED), and any existing requirements documents to ensure a complete understanding of the top-level system missions/functions, environments, Requirements, and imposed Constraints. The MNS defines the needs the system is expected to meet. The CONOPS is developed from the MNS and normally includes an OSED. (The OSED is defined in Paragraph 4.4.4.2.1.) A system understanding from the perspective of these documents ensures that the system's relationship to its environment and external systems is considered during the development of the primary system functions. Figure 4.4-6 is a simplified example of an MNS and CONOPS for an office requiring the capability to record and store information from a computer. This example is used only to develop a sample functional flow diagram (FFD) (Paragraph 4.4.3.2.2.1). An actual MNS and CONOPS include much greater detail.

The system's primary mission(s) is defined using the MNS and any available system

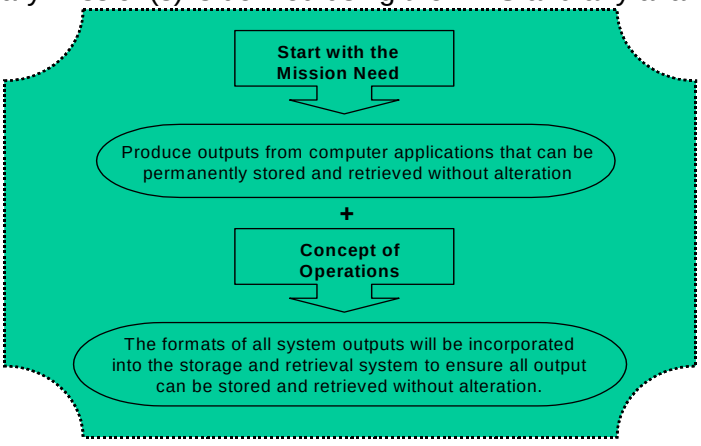


Figure 4.4-6. Mission Need Statement and Concept of Operations

descriptions, such as the OSED, the system's Requirements, and Constraints. This mission(s) is the primary function that the system fulfills, and it is named using the guidelines and naming convention described in the "Introduction to Functional Analysis" (Paragraph 4.4.1).

In addition, the internal and external interfaces (including ambient and operational environments) of the system are identified, and the functional relationships are defined. In the next task, these relationships are depicted through structured analysis using sequence diagrams, FFDs, and N^2 diagrams, which meet nearly all FAA program needs. In these depictions—examples of which appear in Figures 4.4-9 through 4.4-20—a large rectangular box represents the system, and the smaller boxes represent external elements outside of the main system. Flow arrows represent interfaces between the system and the external elements that describe which external element the system is transmitting to/receiving from and what data is being transmitted/received. Figure 4.4-7 shows the standard symbols used in these diagrams. ("Functional Analysis Tools and Techniques" (Paragraph 4.4.5) provides other techniques that may be used with approved tailoring to the process.)

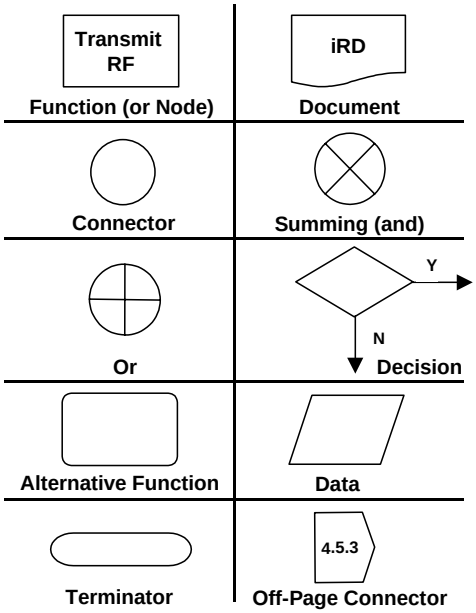


Figure 4.4-7. Symbology Template for Functional Flow Diagramming

In Task 1, the necessary functions that provide the required capabilities of the system, as specified by the need or Requirements, are defined. The activity represented by each of the functions shall be well defined, able to be implemented, and testable; and the interfaces to other functions shall be as simple as possible. It is recommended that these functions be developed with an eye toward the conversion of the Functional Architecture into Requirements and Requirements into a Physical Architecture. The development of complementary Functional and Physical Architectures requires multiple iterations between Functional Analysis, Requirements Management (Section 4.3), and Synthesis (Section 4.5).

Figure 4.4-8 lists functions based on the example MNS and CONOPS depicted in Figure 4.4-6. There are many approaches to describe these functions. The main criterion for task completion is a comprehensive list of the functions the system has to perform in order to meet its mission. For this task, the list does not need to follow a logical order.

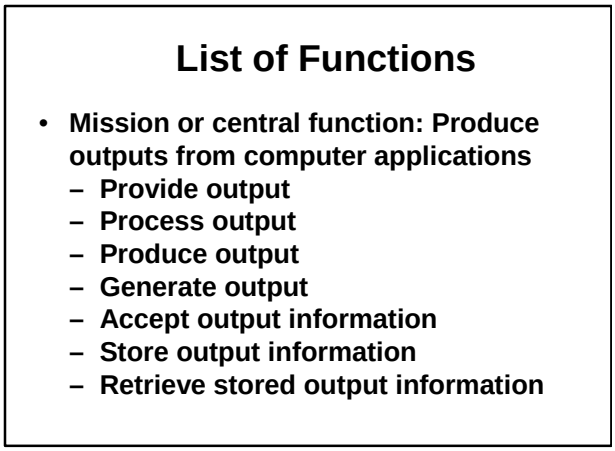


Figure 4.4-8. List of Primary Functions

An analysis of operations and environment may be tailored to represent the available source of information. If detailed references to environmental data are absent in the initial Requirements Documents, Quality Function Deployment or other methods described in Requirements Management may be used as supplements to elicit the information necessary to support follow-on Physical Architecture and Requirements tasks.

Affirmative answers to the following questions signify completion of Task 1:

- Have all missions, phases, and modes of operation been considered for the system?
- Have all functional elements been properly identified?
- Have all functional interfaces of the system to and from the environment been adequately identified and listed (physical/functional interface, connection parameters and modes, etc.)?
- Have the results of this review been captured in a list that identifies the system's mission and primary functions as well as interfaces with other systems and the environment?

4.4.3.2. Task 2: Organize Functions Into Logical Relationships

The function list developed in Task 1 serves as an input to Task 2. The function list includes the central functions required for the system to accomplish its mission, but the list is not necessarily arranged in a sequence or logical relationship. During Task 2, the functions are arranged in at least one of the primary logical flow diagrams, which are applicable to most programs and indicate relationships based on function sequence and/or functional flow (input-function-output). The arrangement of the functions includes independent functions in parallel and dependent functions in series (e.g., when completion of the upstream function is necessary in order to begin the downstream function). A discussion of other techniques (used only when tailoring is approved) is included in "Functional Analysis Tools and Techniques" (Paragraph 4.4.5).

4.4.3.2.1. Sequence Relationships

The sequence family of relationships includes both sequence and timing. Sequence relationships shall be used if sequence or timing is critical to the overall system function and when the relationships are simple. When sequencing is selected, the functions are arranged in order of sequence (i.e., preceding functions depicted before subsequent functions).

4.4.3.2.1.1. Network Diagrams

Sequence relationships may be depicted as network diagrams. These diagrams shall be used if sequence is important to the function operation, but timing is not necessarily critical. Network diagrams display functions and sequential dependencies in a network format. A box (called a node) represents each function, and a line connecting two boxes represents the sequential dependency between the two functions. Figure 4.4-9 depicts a simple network diagram. Some analysts apply an action on line pattern, where the nodes represent events that partition the actions (on the lines) into time frames.

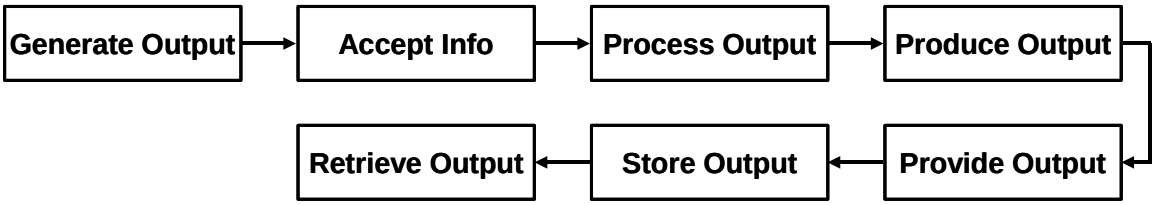


Figure 4.4-9. Depiction of a Sequence Relationship Using a Network Diagram

4.4.3.2.1.2. Time Line Sequence Diagrams

Another way to organize functions in sequence is to use time line sequence diagrams. A time line sequence diagram depicts each function as a line or rectangle on a chart similar to a Gantt chart. The functions are stacked with preceding functions depicted to the upper left of subsequent functions. Time line sequence diagrams shall be used when a sequence relationship is selected, and timing is critical to the function operation. Figure 4.4-10 depicts a simple graphical deterministic time line sequence diagram.

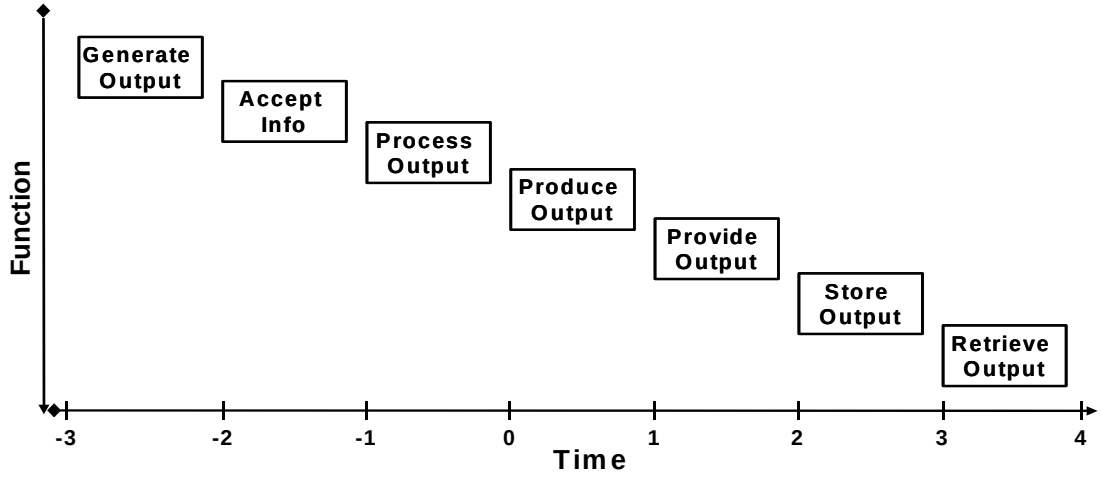


Figure 4.4-10. Depiction of a Sequence Relationship Using a Time Line Sequence Diagram

Time line sequence analysis considers functional durations and provides a more definitive description of the functional sequences than network diagrams are able to convey. It graphically depicts the concurrence, overlap, and sequential relationships of functions and related tasks. Time line sequence analyses are important in the tradeoff process between man and machine, including decisions regarding manual and automatic methods and allocation of times to subfunctions. In addition to defining subsystem/component time requirements, time line sequence analysis is used to develop Trade Studies (Section 4.6) in areas other than time considerations (e.g., is the spacecraft location to be determined by the ground network or by onboard computation using navigation satellite inputs?). Figure 4.4-11 depicts a maintenance time line sheet (TLS) that shows that the availability of an item (distiller) is dependent upon the concurrent completion of numerous maintenance tasks. Furthermore, the figure illustrates the traceability to higher-level requirements by referencing the appropriate FFD.

TIME LINE SHEET | (A) FUNCTION PERFORMANCE PERIODIC MAINT ON VC DISTILLER | (B) LOCATION- ENGINE ROOM 3 | (C) TYPE OF MAINT- SCHEDULED 200 HR PM

(D) SOURCE FFBD 37.5x2 | (E) FUNCTION & TASKS RAS 37.5x17 | (F) TIME - HOURS .5 1.0

TASK SEQ.#	TASK MEMBERS	CREW	
.01	INSPECT COMPRESSOR BELT	A2	.3H
.02	LUBRICATE BLOWDOWN PUMP	B1	.2H
.03	CHECK MOUNTING BOLTS	B1	.1H
.04	CLEAN BREATHER CAP	B1	.1H
.05	CLEAN FOOD STRAINER	C1	.5H
.06	REPLACE OIL	B1	.2H
.07	REPLACE FILTER	C1	.4H
.08	REPLACE V-DRIVE BELT	D1	.9H
.09	CLEAN & INSPECT CONTROL PANEL	C1	.1H
.10	INSTALL NEW DIAPHRAGMS	A1	.7H
.11	CLEAN CONTROLS	B1	.1H

TOTAL MAN-HOURS - - 3.6 MH
ELAPSED TIME - - - - - 1.0 H

Figure 4.4-11. Time Line Sheet for Maintenance of a Distiller

Time line sequence analysis is performed on areas where time is critical to mission success, safety, utilization of resources, minimization of downtime, and/or increasing availability. The following areas are often categorized as time-critical:

- Functions affecting system reaction time
- Mission turnaround time
- Time countdown activities
- Functions requiring time line sequence analysis to determine optimum equipment and/or personnel utilization

Time line sequence analysis supports the development of design requirements for operation, test, and maintenance functions (additional techniques such as mathematical models and computer simulations may be necessary). In addition, the TLS is used to perform and record the analysis of time-critical functions and functional sequences. For time-critical functional sequences, it is necessary to specify the time requirements with associated tolerances.

4.4.3.2.2. Functional Flow (Input-Function-Output) Logical Relationships

The FFD family consists of a group of analyses that depicts functional (input-function-output) relationships between functions. This family includes the Department of Defense standard FFDs, N^2 diagrams, Integrated Definition for Function Modeling (IDEF) techniques, which are described in the following paragraphs, and the Unified Modeling Language (UML), which is described in Paragraph 4.4.5.2.6.2.

4.4.3.2.2.1. Functional Flow Diagrams

The FFD, the FAA's recommended technique for Functional Analysis, is a multi-tier, time-sequenced step-by-step diagram of the system's functional flow. FFDs usually define the detailed, step-by-step operational and support sequences for systems, but they are also used effectively to define processes in developing and producing systems. The software development processes also use FFDs extensively. In the system context, the functional flow steps may include combinations of hardware, software, personnel, facilities, and/or procedures. Although functional flow relationships are more complicated, they also convey more information than sequence diagrams. In the FFD method, the functions are organized and depicted by their logical inputs and outputs. Each function is shown in relation to the other functions by how the inputs and outputs feed and are fed by the other functions. A node labeled with the function name depicts each function. Arrows leading into the function depict inputs, while arrows leading out of the function depict outputs. Figure 4.4-12 depicts the output of function F0 as an input to function F1.

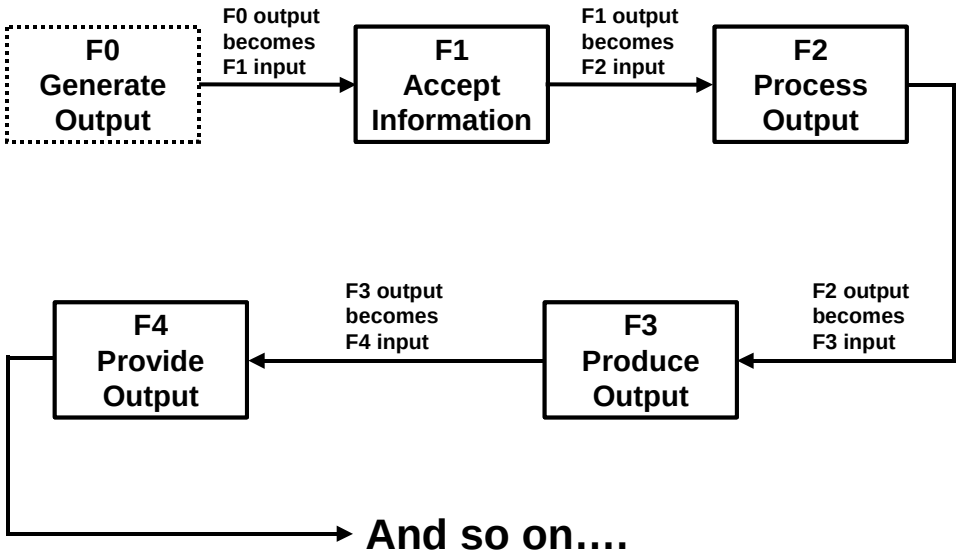


Figure 4.4-12. Functional Flow Relationship

With FFDs, the function is the machine or process that uses inputs to produce outputs. To illustrate: If a turbine engine is the system, then the function is the conversion of oxygen and fuel into mechanical energy in the form of thrust or torque. In Figure 4.4-13, the function is depicted as a box (node) with a function title of "Generate Torque/Thrust." Inputs are the elements needed for the function to operate correctly; the production of mechanical energy using a turbine engine requires oxygen and fuel. Therefore, oxygen and fuel are inputs to that function. Inputs are depicted as arrows leading into the functional node with the input arrows labeled appropriately. The output is the product of the function. In Figure 4.4-13, the engine function generates the "Torque/Thrust." The output is depicted as the arrow leading out of the functional node with the output arrow labeled appropriately.

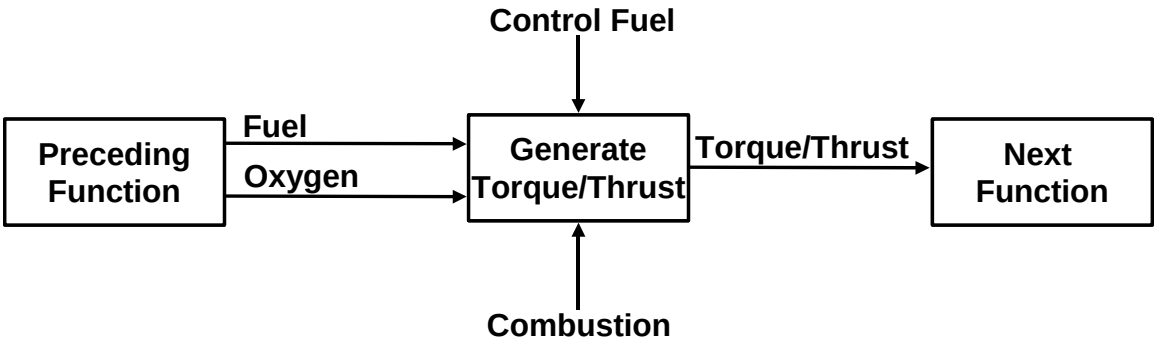


Figure 4.4-13. Input-Function-Output Relationship

NOTE

When IDEF techniques (Paragraph 4.4.3.2.2.3) are being used, controls and mechanisms may be added. Controls are elements that manage the function and

are outputs of other functions, while mechanisms are elements that enable the function to work. Controls are depicted as arrows entering the function from the top, and mechanisms are depicted as arrows entering the function from the bottom.

The functions depicted so far have been serial functions; however, many functions are parallel (i.e., they are functions that (1) independently feed the same downstream function, and/or (2) occur simultaneously). Figure 4.4-14 illustrates parallel functions.

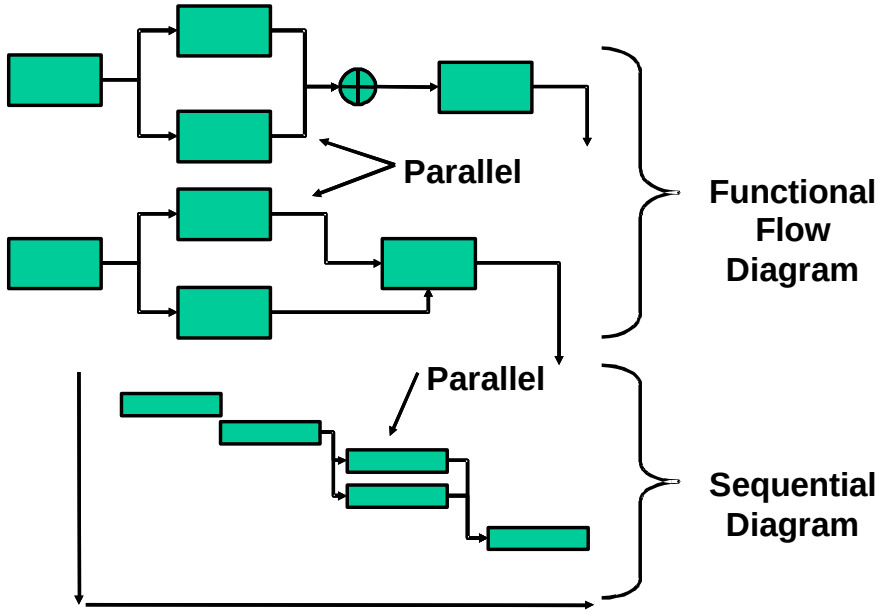


Figure 4.4-14. Parallel Functions

Figure 4.4-15 depicts a functional flow organization of the functions, in which functions are broken down into subfunctions using the techniques described in Task 2. This figure represents a simplified FFD, in which the inputs and outputs are not labeled, and the controls and mechanisms are not identified. Functional Analysis is performed to the level of detail needed to depict the functional description of the system.

Figure 4.4-15 also shows multiple functional levels; however, only the top level is complete. Each lower level shows an example expansion of one function. For example, at the second level, the top-level function F1 is expanded into its second-level functions, F1.1 through F1.6. At the third level, second-level function F1.4 is expanded. Finally, at the fourth level, function F1.4.3 is expanded. Each level indicates a different example of typical functional flow paths. Usually, only one or two levels are shown in one diagram to avoid confusion.

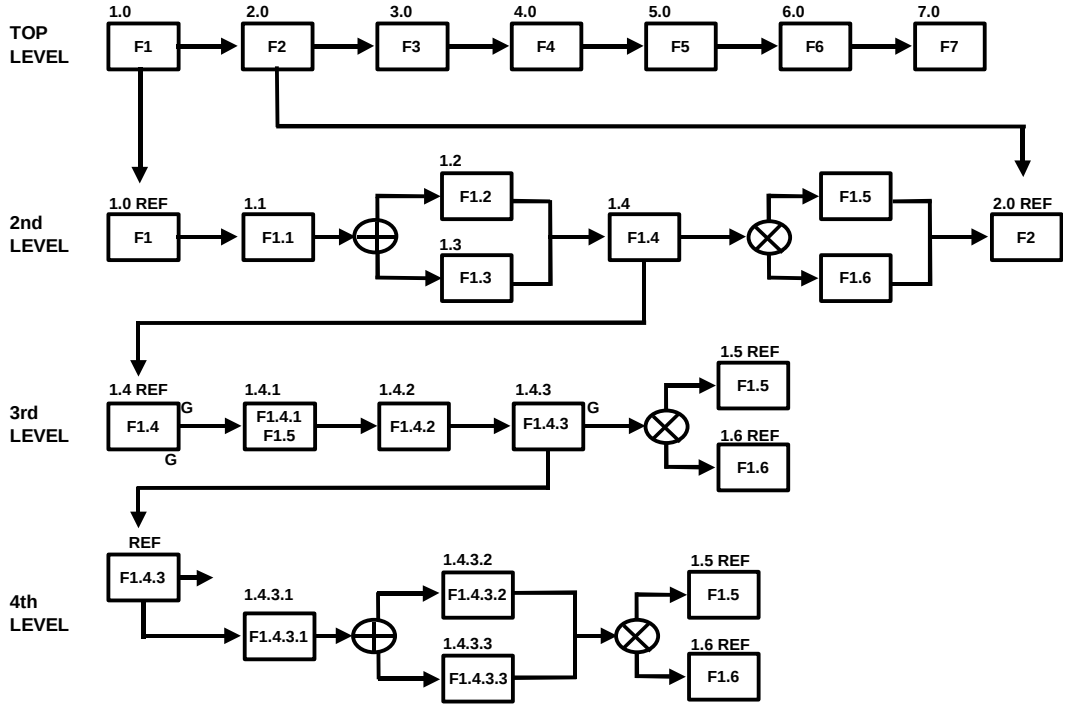


Figure 4.4-15. Generic Functional Flow Diagram Example

Adherence to the following rules promotes common understanding of FFDs:

- Number top-level functions with even integers and zero decimals (e.g., 1.0, 2.0, etc.) and cover the complete span of anticipated lifecycle functions
- Depict inputs to functions as entering from the left side and outputs as leaving from the right side
- Depict mechanisms as entering from the bottom and controls as entering from the top
- Display lower-level functions as emanating from the bottom
- Define the name of the function inside the box, replacing F1, F2, etc.
- Indicate a reference function (ref) at the beginning and end of all functional sequences, *except at the top level*
- Use an “OR” gate to indicate alternative functions; use an “AND” gate to indicate summing functions, where all functions are required (Figures 4.4-7 and 4.4-16)
- Indicate a “GO” “NO GO” sequence with an arrow leaving the right side of the function with the letter “G” for “GO” and an arrow out the bottom with “G-bar” for “NO GO”
- As is customary, when the second level or lower level is shown on a separate page, list the title of the function at the top center of the page for reference
- Typically, do not show the information flow, content of each functional step, and timing details on FFDs

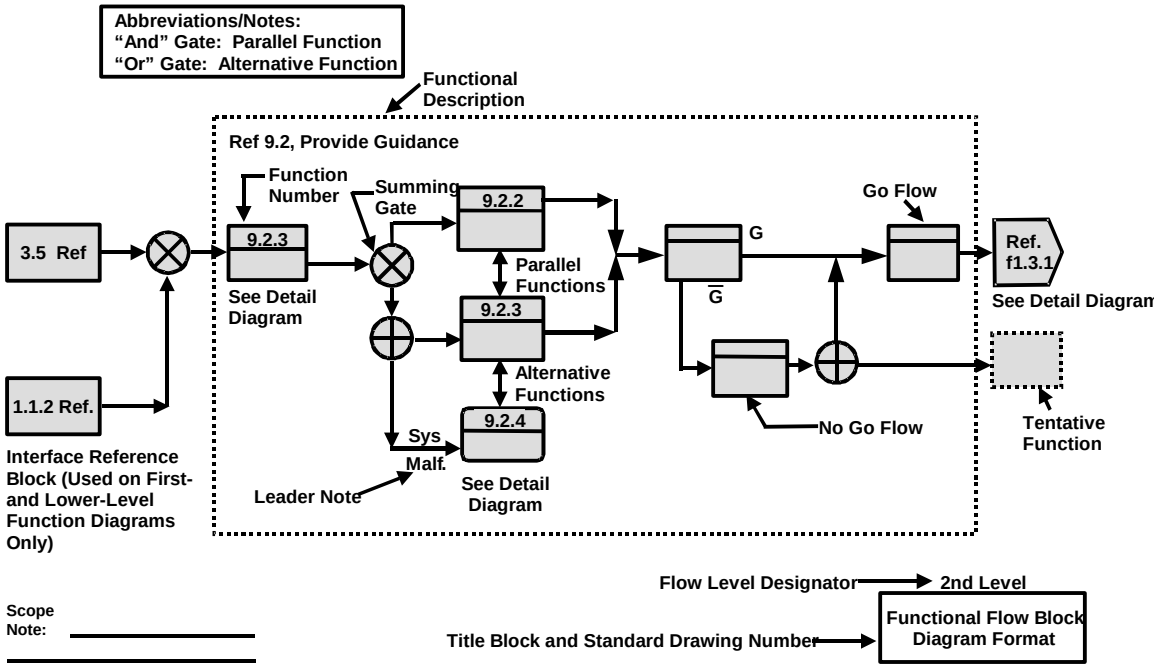


Figure 4.4-16. Functional Flow Diagram Example

4.4.3.2.2.2. Functional N^2 Diagrams

The N^2 diagram is a systematic approach to identify, define, tabulate, design, and analyze functional and physical interfaces. A functional N^2 diagram depicts the interfaces between functions in a system. The N^2 diagram is a visual matrix that requires the user to generate complete definitions of the system functional interfaces in a rigid, bidirectional, fixed framework. A basic N^2 diagram is illustrated in Figure 4.4-17.

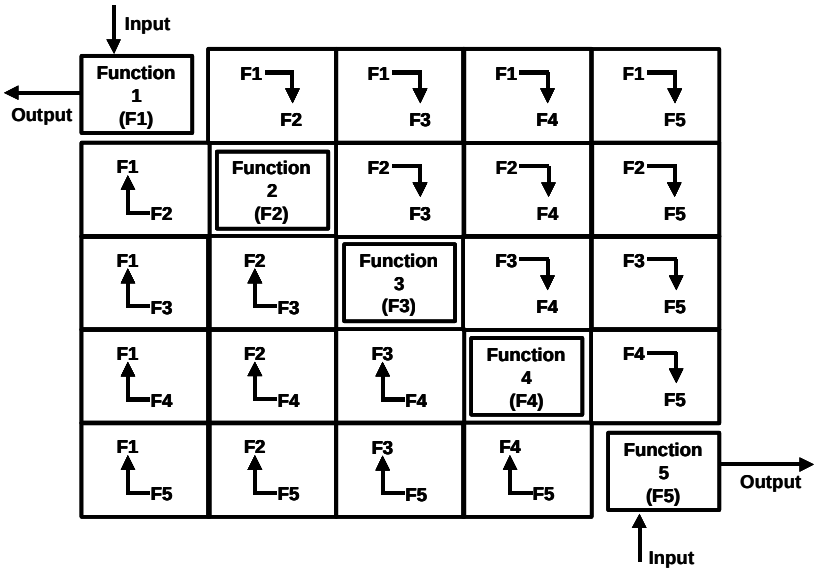


Figure 4.4-17. Functional N^2 Diagram

The N^2 diagram customarily is used to develop data interfaces, primarily in the software areas; however, it also may be used to develop other interfaces, including functional and physical interfaces. In this method, the system functions are placed on the diagonal axis; the remainder

of the squares in the N x N matrix represents the interface inputs and outputs. The presence of a blank square indicates that there is no interface between the respective system functions. Data flows in a clockwise direction between functions (i.e., the symbol F1 ↴ F2 indicates data flowing from function F1 to function F2; the symbol F2 ↥ F1 indicates the feedback). The transmitted data is defined in the appropriate squares. The diagram is complete when each function has been compared to all other functions. The N^2 diagram may be used in successively lower levels down to the component functional level.

N^2 diagrams are a valuable tool for not only identifying functional interfaces, but also for pinpointing areas where conflicts may arise between functions so that system integration proceeds smoothly and efficiently.

4.4.3.2.2.3. Integrated Definition for Function Modeling Diagrams

IDEF is a common modeling tool for conducting analysis, development, and integration of information technology systems and software engineering analysis. Whereas FFDs show the functional flow of a product, IDEF diagrams show:

- Data flow
- System control
- Flow of lifecycle processes

The U.S. Air Force originally developed IDEF for manufacturing planning. IDEF is a compound acronym that stands for Integrated Computer-Aided Manufacturing Definition. Originally called IDEF, other IDEF languages have since been developed, forcing the languages to adopt numbering system; thus, this technique is now called IDEF 0. IDEF 0 has demonstrated an ability to depict a variety of engineering, operational, manufacturing, and other types of processes at any level of detail. It provides disciplined, rigorous, and precise descriptions while promoting standardization in use and interpretation.

IDEF is a model composed of a hierarchical series of diagrams, text, and a glossary that are cross-referenced. The two primary modeling components are functions and data objects that interrelate the functions. As shown in Figure 4.4-18 (IDEF box format), the position at which the arrow attaches to a box conveys the role of the data object interface. These roles consist of:

- Input
- Mechanism
- Output
- Control

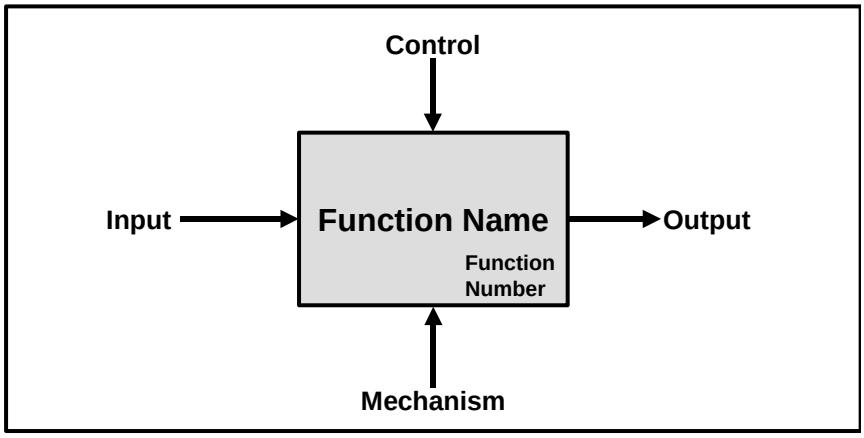


Figure 4.4-18. Integrated Definition for Function Modeling Function Diagram

The inputs, the data objects acted upon by the function or operation, enter from the left. The mechanism (additional support to perform the function) arrow attaches to the box from the bottom. The outputs of the function leave the function box from the right. The controls enter the top of the box.

The IDEF process begins with the identification of the prime function to be decomposed. This function is identified on a top-level context diagram that defines the scope of the particular IDEF analysis. Figure 4.4-19 illustrates a top-level context diagram for an information system management process. From this diagram, lower-level diagrams are generated. An example of a derived diagram—called a “child” in IDEF terminology—for a lifecycle function is shown in Figure 4.4-20.

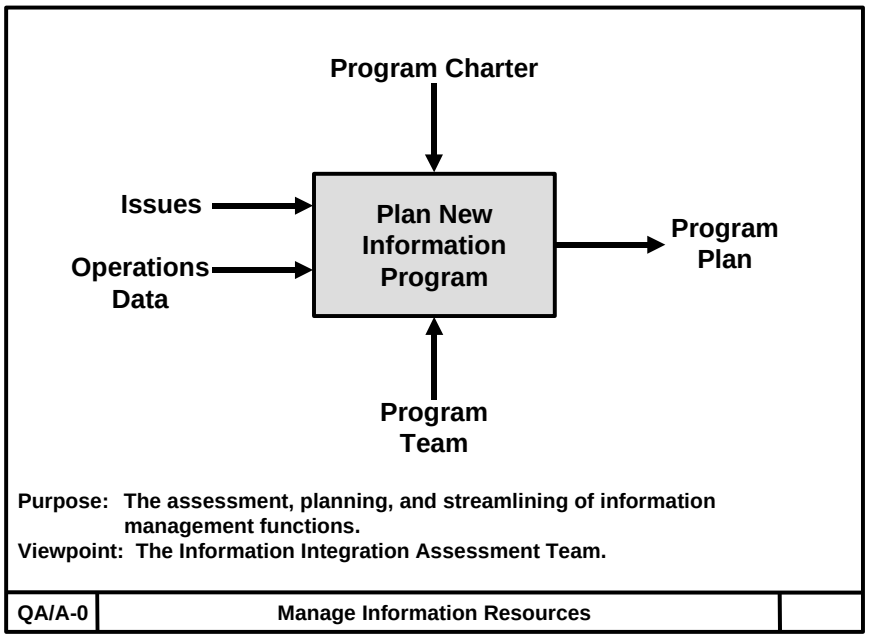


Figure 4.4-19. Top-Level Context Diagram

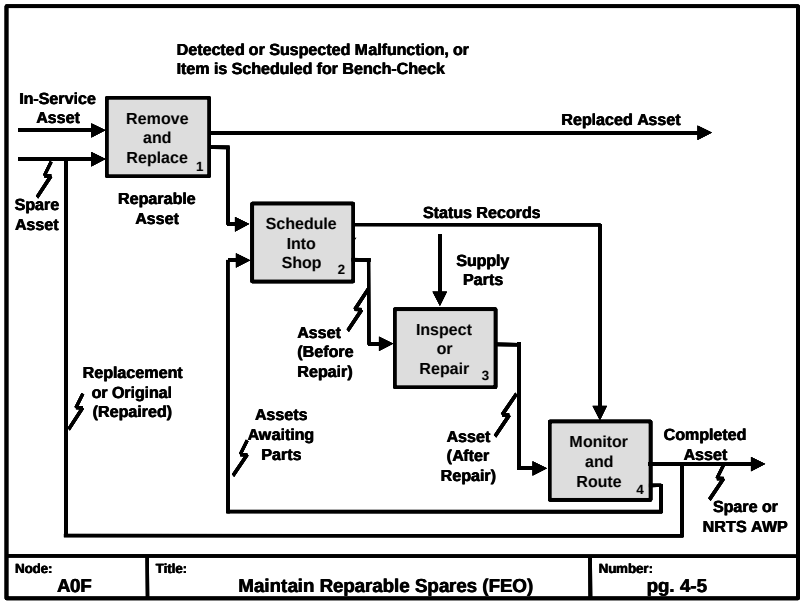


Figure 4.4-20. Derived Diagram ("Child" in Integrated Definition for Function Modeling Terminology)

Affirmative answers to the following questions signify completion of Task 2:

- Are all functions in the function list depicted?
- Are all functions written in the form verb–noun format?

- Are all functional interfaces depicted graphically?
- Does the depiction show end-to-end functional relationships?
- Are parallel and serial relationships accurately depicted?

4.4.3.3. Task 3: Decompose Higher-Level Functions Into Lower-Level Functions

In this task, higher-level functions are decomposed into subfunctions, with specificity increasing at each level of decomposition. Functional decomposition is performed using the techniques described in Tasks 1 and 2 with respect to sequence and logical diagramming or alternatively with the techniques described in "Functional Analysis Tools and Techniques" (Paragraph 4.4.5). The stepwise decomposition of a system basically is a top-down approach to problem-solving. Shown graphically in Figures 4.4-21 through 4.4-24, the decomposition is carried to a level at which the functions have been totally decomposed into basic subfunctions, and each subfunction at the lowest level is completely, simply, and uniquely defined by its Requirements. This means that functional decomposition continues as long as there is a further need to define lower-level Requirements. When the requirements development process ceases, Functional Analysis may cease.

The objective of Task 3 is to develop a hierarchy of Functional Analysis diagrams that describes the functions at all levels of the system. This hierarchy is only a portion of the Functional Architecture, which is not complete until all Requirements and other Constraints have been appropriately decomposed.

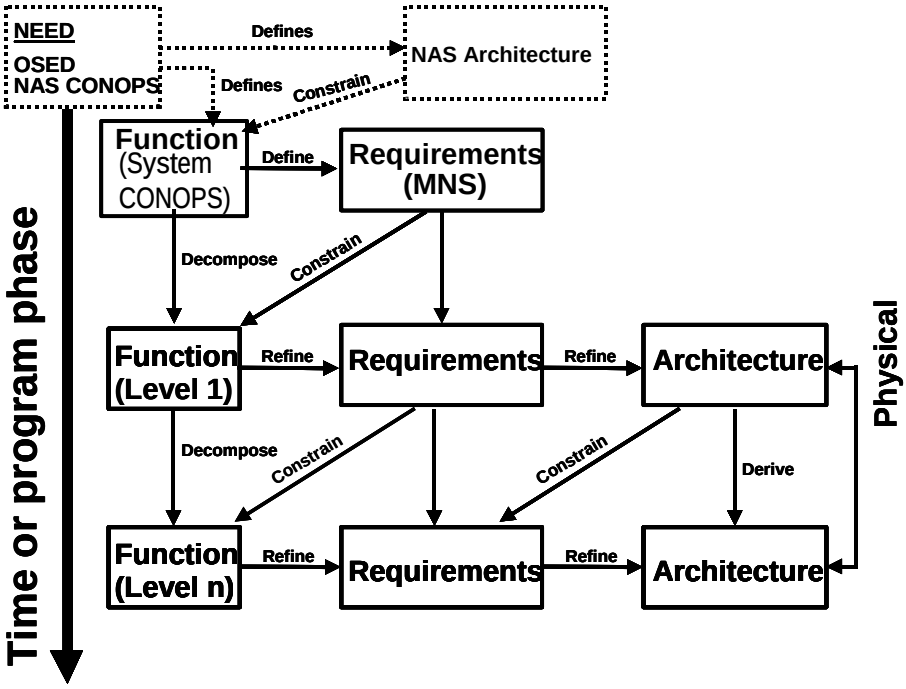


Figure 4.4-21. Decomposition of Higher-Level Functions into Lower-Level Functions

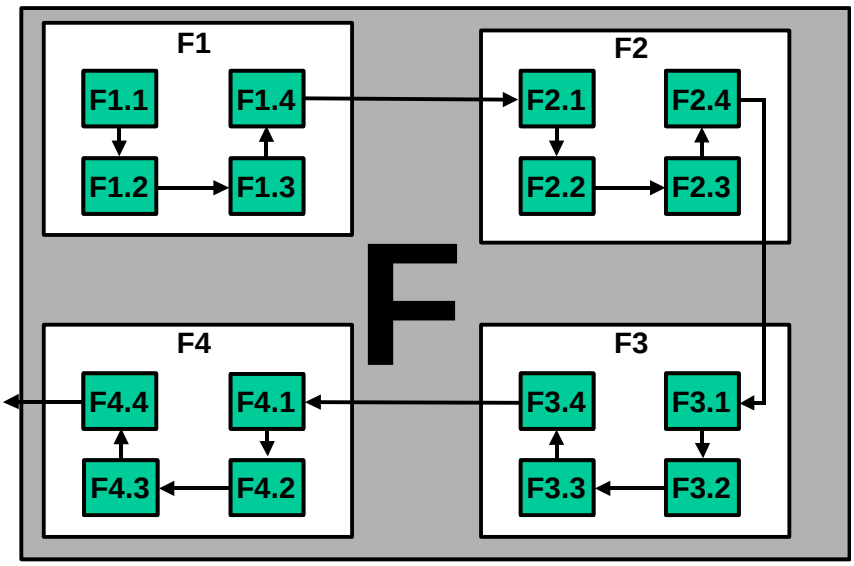


Figure 4.4-22. Another View of Decomposition of Higher-Level Functions into Lower-Level Functions

Task 3 is performed iteratively using the steps and techniques described in Tasks 1 and 2. Since higher-level functions exist for this task, the subfunctions are based on the higher-level functions developed in the previous tasks. In Figure 4.4-23, which uses the functions list from Figure 4.4-8, function F3 is decomposed into subfunctions labeled as the second level. Next, the functions in the second level are further decomposed to the third level. This process continues until all the functions are totally decomposed into basic subfunctions, and each subfunction at the lowest level is completely, simply, and uniquely defined by its Requirements. At each level, Functional Analysis feeds Requirements Management (Section 4.3) and Requirements feeds Synthesis (Section 4.5), as shown in Figure 4.4-21 and further illustrated in Figures 4.4-24 through 4.4-27.

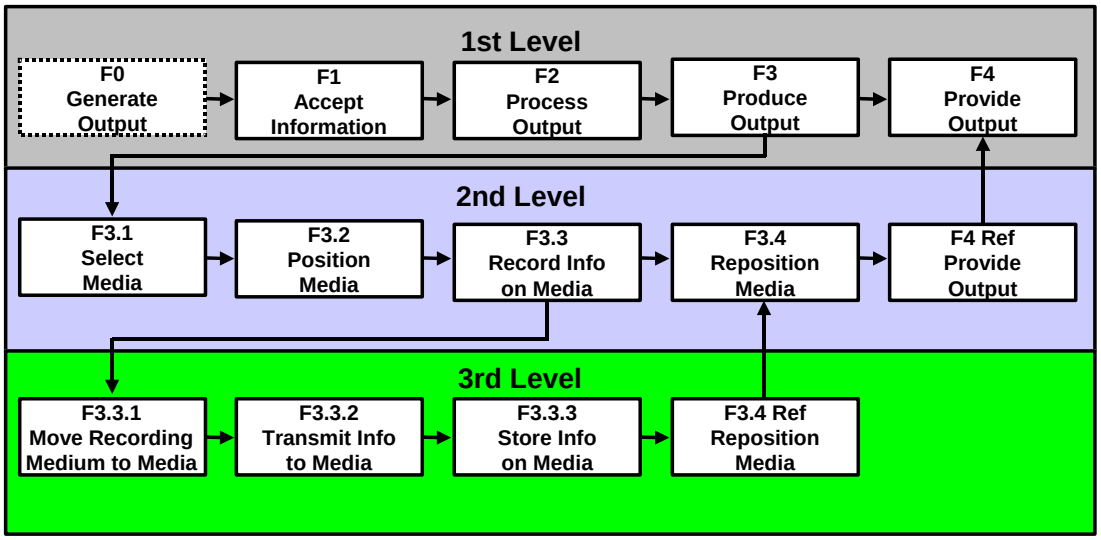
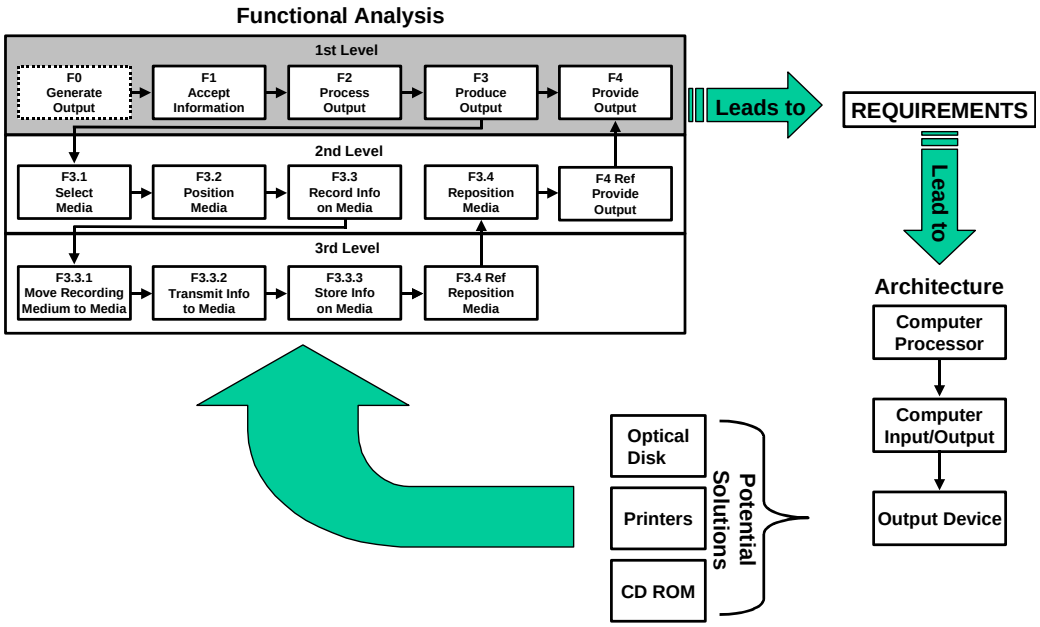
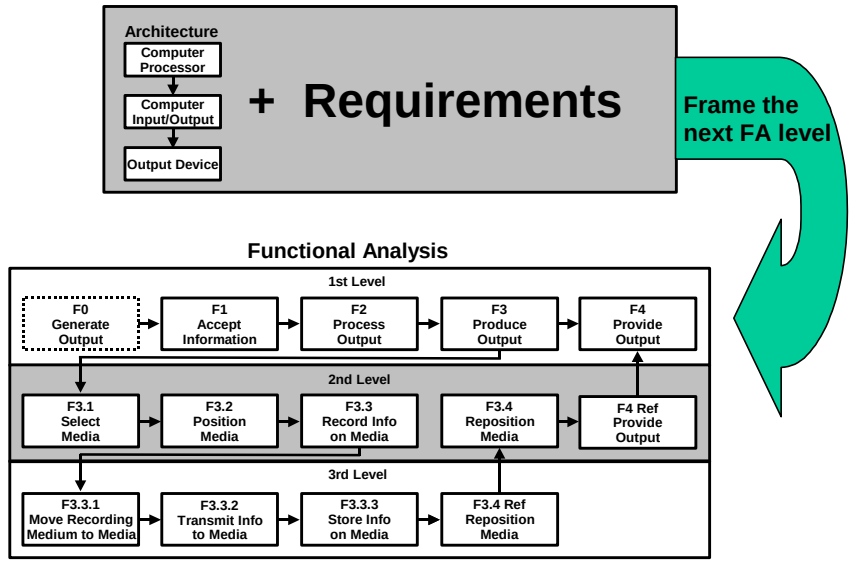


Figure 4.4-23. Higher-Level Functions Broken Down into Lower-Level Subfunctions



466 **Figure 4.4-24. Functions Lead to Requirements; Requirements Lead to Physical Architectures**

467 Requirements Management and Synthesis detail the process that turns functions into
468 Requirements and Requirements into a Physical Architecture. It is important to note that the
469 next Functional Analysis level is bound and framed by the Requirements and Physical
470 Architecture refined from the preceding Requirements Management and Synthesis activities
471 (Figure 4.4-25).



472

473 **Figure 4.4-25. Requirements and Physical Architecture Frame the Next Functional Analysis Level**

474 When this spiral process completes one rotation, the Functional Analysis process recommences
475 (Figures 4.4-26 and 4.4-27) at the next lower level and repeats until each function is totally
476 decomposed into its basic subfunctions, and each subfunction at the lowest level is completely,
477 simply, and uniquely defined by its Requirements.

Affirmative answers to the following questions signify completion of Task 3:

- Has a complete set of Functional Analysis diagrams been prepared?
- Has each function been decomposed to its lowest level within program needs?
- Is each function completely, simply, and uniquely defined by its Requirements?
- Has a description of each function been developed?
- Is the requirements development complete?

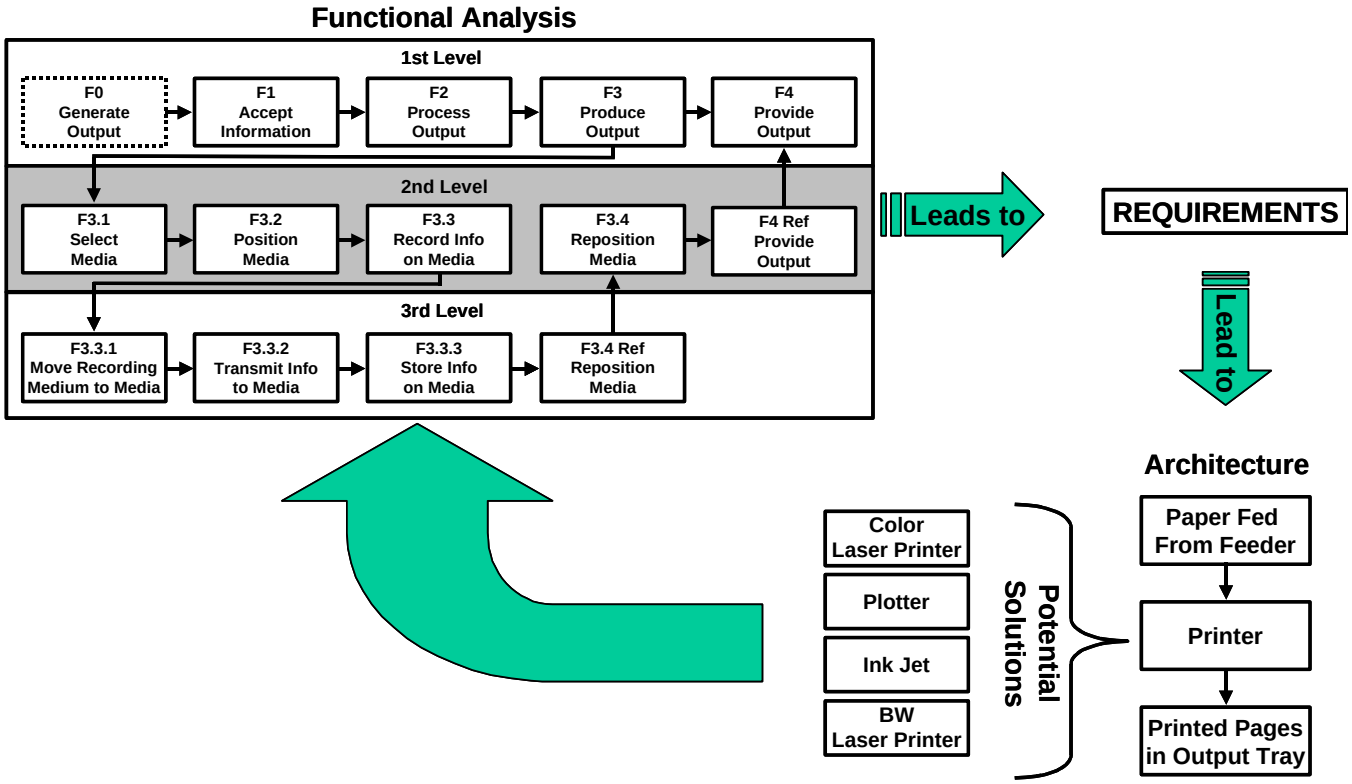


Figure 4.4-26. Repeating the Functional Analysis Process at the Next Lower Level

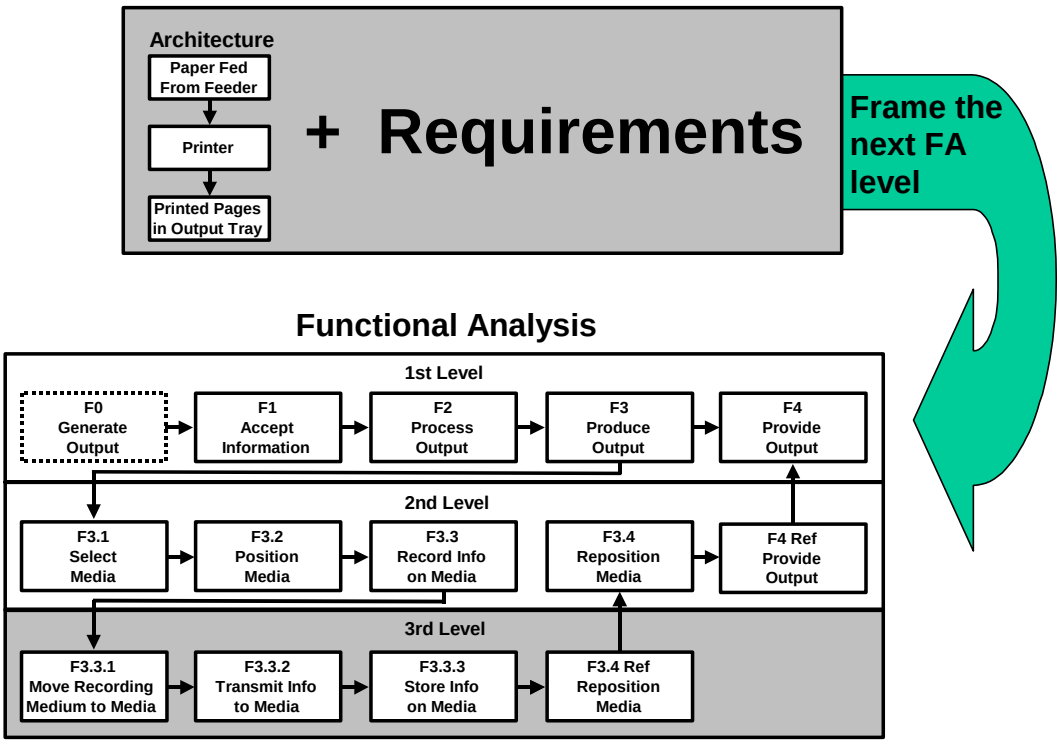


Figure 4.4-27. Preceding Requirements and Physical Architectures Continue To Frame Next Lower Level

4.4.3.4. Task 4: Evaluate Alternative Decompositions

In this task, alternative decompositions of functions (Functional Architectures) and Requirements at all levels are evaluated. These evaluations are necessary because there is no single "correct" decomposition; however, not all decompositions are of equal merit. It is necessary to evaluate alternative decompositions in order to select the decomposition best suited to the Requirements. There are other reasons to evaluate alternative decompositions. For example, as a result of Synthesis there may surface design constraints such as the desire to use commercial-off-the-shelf (COTS) or non-developmental item (NDI) components. Multiple Functional Architectures may then be produced to accommodate alternatives in accordance with various combinations of constraints. These are then compared using the Trade Studies process (Section 4.6) with the design criteria from Synthesis in order to select the Functional Architecture that most effectively meets mission objectives.

The evaluation of alternative decompositions of functions is subjective and dependent upon personal preference and taste. The purpose of Task 4 is to ensure that other methods to conduct the decomposition are evaluated. In this task, personal preference and consensus among the stakeholders become factors in the selection of the best Functional Architecture. Any selected Functional Architecture shall reflect the system's functions. However, variances in the alternative Functional Architectures may provide a competitive edge to one or more of the alternatives.

By the end of this process, the Requirements for each subfunction at the lowest levels of the Functional Architecture are allocated via the Synthesis process ("Task 3: Allocate Requirements" (Paragraph 4.3.4.3)) to hardware, software, interfaces, operations, or a

database, and then to a specific configuration item. As it is necessary to verify Requirements, the objective of Task 4 is to select those decompositions that promote straightforward Requirements that may be validated and verified. (Validation and Verification (Section 4.12) further addresses this issue.) In addition, decompositions that allow a single function to be used at several places within the hierarchy, thereby simplifying development, may be identified.

Task 4 requires "best engineering judgment," as the "goodness" of each functional decomposition is evaluated by measuring the degree to which each module displays the following attributes:

- Performs a single function
- Is a logical task
- Leads to a Requirement(s) that may be separately validated
- Has a single input point and a single output point
- Is independent within each level of the hierarchy (higher independence allows the implementation of the module independent of the other modules)

Because system design does not occur in a vacuum, it is necessary to consider opportunities to use COTS or NDI hardware and software. As a result, a subfunction that has already been implemented in a compatible form on another system may be preferred to one that has not.

The selection of a final system functional decomposition signifies completion of Task 4.

4.4.3.5. Task 5: Document Functional Analysis Baseline

The last task in the Functional Analysis process is documenting the process, including the selected Baseline as the basis for Requirements Management (Section 4.3) and Synthesis (Section 4.5). The documentation includes the outputs listed in Figure 4.4-1. At this point, any necessary revisions or changes to the functional decomposition, sequence and time lines, functional interfaces, etc., are made to ensure their completeness and consistency with one another. Also, the products of the Functional Analysis process (e.g., FFDs, functional descriptions, function interface descriptions, and time lines) are developed. These products may be documented separately or combined in a Functional Analysis Document (FAD).

Affirmative answers to the following questions signify completion of Task 5:

- Have all of the initial functions been decomposed into subfunctions?
- Do the subfunctions cover the total scope of the parent function?
- Are the functions arranged correctly with respect to the dependence of the functions?
- Have all functional interfaces been defined?
- Have any new functional interfaces between initial functions been identified that were discovered during the function decomposition process? (These may drive new system element interfaces.) If so, have the new interfaces been documented in control sheets?
- Has a Functional Analysis document been prepared to document the functional Baseline?
- Have all functional Requirements been identified and decomposed?

4.4.4. Outputs of Functional Analysis

4.4.4.1. Functional Architecture

The most common output of the Functional Analysis process is a "living" Functional Architecture document that contains a tailored combination of the following:

- Functional Architecture Baseline
- Functional interface list
- Alternative decompositions
- FFDs
- Functional time lines and sequences
- Data flow diagrams (DFD) and threads
- N^2 diagram
- Other functional descriptions

4.4.4.2. Concept of Operations

In addition to the previous list, the CONOPS may also be an output of the Functional Analysis process. A CONOPS is a user-oriented document that describes system functional characteristics for a proposed system from the user's viewpoint; it is essentially a top-level narrative Functional Analysis. It explains the existing system, current environment, users, the interaction among users and the system, and organizational impacts. The CONOPS document is written in order to communicate overall quantitative and qualitative system characteristics to the user, buyer, developer, and other organizational elements. The CONOPS aids in requirements capture and communication of need to the developing organization. Posing the need in the user's language helps to ensure that the user is able to more accurately express the problem. Subsequently, the system engineers have a better foundation upon which to begin the lower-level Functional Analyses, requirements definition, and initial design of the system.

Not all CONOPS are written as functional analysis documents. In these cases, the CONOPS would be an input to Functional Analysis rather than an output.

The following is a list of the essential elements indicative of all CONOPS:

- Description of the current system or situation
- Insight into the user's environment
- Description of the functions to be performed
- Description of the needs that motivate development of a new system or modification of an existing system
- Insight into the new Requirements
- Opportunity for the developer to recommend alternative solutions
- Description of the operational features of the proposed system
- User's view of the Requirements

At minimum, there are two levels of CONOPS: (1) NAS Level and (2) System Level CONOPS.

A NAS Level CONOPS is performed at the highest level of the Functional Analysis process and is a narrative expression of the user's desired change with some performance indicators. It is a high-level document that indicates, from the user's perspective, the desired end-state for the respective system in the NAS.

A System Level CONOPS is an extension of a NAS Level CONOPS with an emphasis on a particular system. It is more detailed and substantial, but it is still an expression of the user's needs with respect to a specific system within the NAS. It is recommended that a System Level CONOPS, in particular, have the following characteristics:

- Written in the user's language using the user's preferred format
- Written in narrative prose (in contrast to a technical requirements specification)
- Organized so as to tell a story and accompanied by visual forms (diagrams, illustrations, graphs) and storyboards whenever possible
- Provide a bridge between the user's needs and the developer's technical requirements documents
- Describe the user's general system goals, mission, function, and components
- Evoke the user's views and expectations
- Provide an outlet for the user's preferences
- Provide a place to document vague and unmeasurable Requirements (i.e., the user is able to state his/her desire for fast response or reliable operation); these desires are quantified during the process of developing the requirements specifications and during the flowdown of Requirements to the Physical Architecture
- Make the user feel in control

Figure 4.4-28 serves as a guideline for System Level CONOPS content.

1. Introduction
   a. System overview
   b. Definition of terms
   c. References
2. Operational need
   a. Operational problems solved
   b. Opportunities created
   c. Existing operations/functions that require change
   d. Organization constraints
   e. Actors that will interact with system
3. System justification
   a. Capability shortfalls of current system
   b. Potential benefits of new system
   c. Identified priorities of new features
      i. Critical
      ii. Essential
      iii. Routine
   d. Assumptions and constraints
4. OSED (include if available)
5. Business impact
   a. Impact on current business operations
   b. Changes to organization

Figure 4.4-28. Content Format for a System Level Concept of Operations

4.4.4.2.1. Operational Services and Environmental Description

The OSED is a comprehensive, holistic Communications, Navigation, and Surveillance/Air Traffic Management system description. It describes the services, environment, functions, and mechanizations that form a system's characteristics.

"What Is a System?" A system (as defined in Section 2.1) is:

> *An integrated set of constituent pieces that are combined in an operational or support environment to accomplish a defined objective. These pieces include people, hardware, software, firmware, information, procedures, facilities, services, and other support facets.*

The 5M Model, illustrated in Figure 4.4-29, graphically represents this system view. Useful system descriptions exhibit two essential characteristics: correctness and completeness. Correctness in a system description means that the description accurately reflects the system with an absence of ambiguity or error in its attributes. Completeness means that no attributes have been omitted and that the attributes stated are essential and appropriate to the level of

detail called for in the description. System descriptions that include all 5M Model elements achieve these two characteristics.

The 5M Model states that there are five basic integrated elements in any system. These elements are (1) the functions that the system needs to perform; (2) the human operators and maintainers; (3) the equipment used in the system, composed of the hardware and software; (4) the procedures and policies that govern the system's behavior; and (5) the environment in which it is operated and maintained.

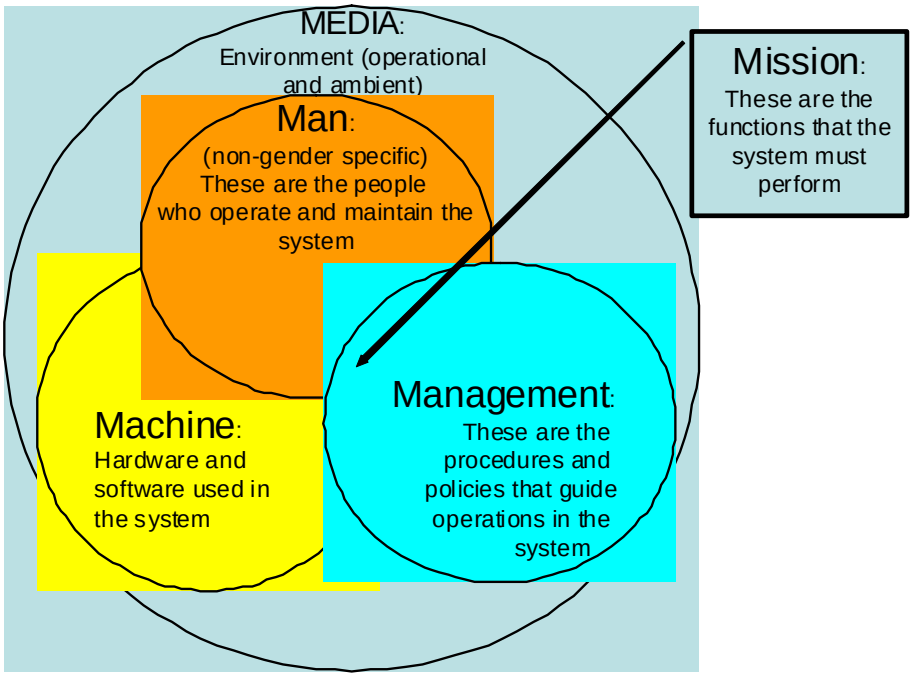


Figure 4.4-29. 5M Model

RTCA/DO-264, Annex C, contains detailed guidelines for the OSED for use as a starting point. For the purposes of SE in the FAA, these guidelines were tailored. It is recommended that an OSED have, at minimum, the information in Figure 4.4-30.

1. Operation Service Description: This section of the OSED is used to summarily describe the air traffic services and operational context of the new capability. This section describes the new air traffic service from an operator's viewpoint.
2. Functional description or architecture: This section describes the functions and Functional Architecture in accordance with Functional Analysis.
3. Procedures: This section describes the existing and new procedures and policies that govern the system's operation or maintenance and includes:
   a. Operational requirements and regulations, including separation minima
   b. Deployment requirements
   c. Operational scenarios
4. Human elements of the system: This section describes the operators and maintainers of the system, including information regarding:
   a. Anthropometric requirements
   b. Training requirements
   c. Specific skill set requirements
   d. Human-system integration requirements
5. Equipment and software: This section describes any known hardware and software that is required for system operation. This section, in particular, may not be appropriate in the early stages of development.
6. Environment description: This section is an expression of the various conditions in which the system is operated, including:
   a. Operational: The operational environment includes factors, such as traffic density and flow, flight phases, traffic complexity, route configuration, type of control, use of visual or instrument flight rules, etc.
   b. Ambient: The ambient conditions refer to visual and instrument meteorological conditions, altitudes, terrain elevations, and physical conditions, such as electromagnetic environment effects, precipitation, icing, etc.
7. Nonfunctional requirements: This section describes any other Requirements that are not covered in the other sections and includes, but is not limited to, the following:
   a. Time constraints
   b. Information exchanges
   c. Exception handling

Figure 4.4-30. Guidelines for an Operational Services and Environmental Description

4.4.4.3. Concerns/Issues

Appendix D contains guidance on concerns/issues as a product of Functional Analysis.

4.4.4.4. Tools/Analysis Requirements

Tools/Analysis Requirements for performing Functional Analysis throughout the remainder of the program's lifecycle needs to be provided to the Integrity of Analyses process (Section 4.9).

4.4.4.5. Planning Criteria

Any Planning Criteria necessary for performing Functional Analysis throughout the remainder of the program's lifecycle needs to be provided to the Integrated Technical Planning process (Section 4.2).

4.4.4.6. Constraints

Constraints on trade studies that surface as a result of performing Functional Analysis are to be provided to the Trade Studies process (Section 4.6).

4.4.5. Functional Analysis Tools and Techniques

Contractors working for the FAA may choose to employ structured analysis models rather than the FFDs preferred by FAA. To facilitate communication between FAA system engineers and these contractors, it is recommended that FAA system engineers understand these models in order to engage in technical conversations with contractors who employ them.

4.4.5.1. Tools

Analysis tools may include but are not limited to general SE and design/simulation aids. Because requirements represent the basic thread through SE, Functional Analysis data shall be interoperable with requirements definition information. The results of the Functional Analysis process shall be captured in order to modify system requirements and other derived products.

The selection of tools shall ensure that the data is transportable and able to be integrated with other related Functional Analysis results. A list of tools that may be used to perform Functional Analysis is available on the International Council on System Engineering Web site (www.incose.org).

4.4.5.2. Techniques

In addition to the techniques described in "Task 2: Organize Functions Into Logical Relationships" (Paragraph 4.4.3.2), this paragraph covers several alternative approaches that FAA system engineers may come in contact with from organizations that apply techniques other than FFDs. These techniques are provided in order to cover two issues: (1) cases in which time line sequence diagrams and FFDs do not adequately address FAA needs; and (2) cases in which contractors use these techniques to perform Functional Analysis, and the FAA engineers need to understand what they mean. The alternatives include the following:

- Hierarchical functional block diagramming
- Modern structured analysis
- Models and simulation
- Thread analysis
- Object-oriented analysis (OOA)

4.4.5.2.1. Hierarchical Functional Block Diagramming

By listing the functions for an expansion of one block on a higher-tier block diagram, the engineer is actually engaging in function outlining, which is equivalent to hierarchical block diagramming. Rather than building sequences of functions in FFDs, the engineer may build an indentured list or hierarchy of functions where, in order to accomplish a particular function, it is necessary to complete the immediate lower-tier functions first.

Generally, this process is easier to follow than a sequence-oriented model. Thus, the question arises: Why build a sequence- or flow-oriented diagram when the goal is a hierarchical physical architecture? It is easier for system engineers to move from a hierarchical functional diagram to a hierarchical architecture diagram than from a sequence-oriented functional diagram to a hierarchical physical architecture diagram. However, therein lies the problem. Engineers employing hierarchical functional diagrams often define a functional architecture based on the last physical architecture they worked on, which generally causes a one-to-one correspondence between their functional and physical architectures. The danger in this approach is the potential for the engineer to fail to consider all of the alternative implementations of the needed functionality and, subsequently, miss opportunities to take advantage of new technology. It is more difficult to move from a sequence-oriented functional model to a hierarchical physical architecture diagram; however, that difficulty encourages a more comprehensive examination of methods to implement exposed functionality.

4.4.5.2.2. Modern Structured Analysis

Modern structured analysis offers a more free-form analytical environment than the block-oriented models. The modern structured analysis model is constructed using DFDs that feature bubbles rather than blocks, a data dictionary (DD), and process specifications (p-spec).

4.4.5.2.2.1. Data/Control Flow Diagrams and Context Diagrams

Data/control flow diagrams (D/CFD) graphically model the processes that transform data/control in a system. These diagrams model the system's work as a network of activities that accept and produce data/control messages. Alternatively, they are also used to model the system's network of activities as work accomplished on a processor. Each successive level of D/CFDs represents the internal model of the transformations contained in the previous level of D/CFDs.

The context DFD—the ultimate DFD—consists of one bubble depicting the system connected to terminators drawn as blocks and named to identify the external inputs and outputs of the system. The bubble of the context DFD is decomposed to expose more detailed needs of the system. The lower-tier DFDs, of which there may be hundreds or even thousands for a complex problem, consist of only four objects:

- Bubbles (drawn as simple circles) identify needed computer processing. Bubbles have functionality that may be further decomposed in a lower-tier DFD or defined in a p-spec. P-specs are written only for the lowest-tier bubbles in the diagrams. Needed product behavior may be explored and illustrated by structured English, tables, or state diagrams within the p-spec.
- Directed line segments (arrows) show the flow of data between the bubble and temporary data stores.
- Temporary data stores (represented by a pair of parallel lines) identify a need to temporarily store data created in a bubble or applied from outside the system.
- Data sources and external inputs are represented by rectangles.

Figure 4.4-31 illustrates the application of DFDs and the top-down decomposition process used to produce a system model.

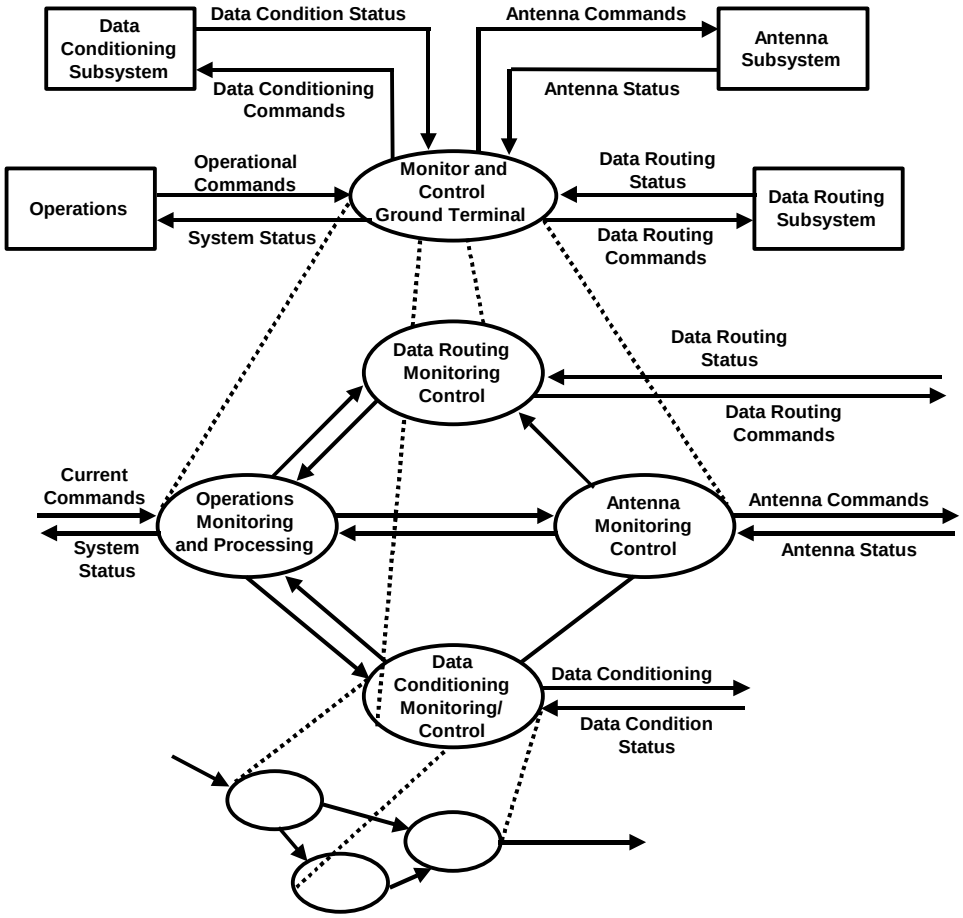


Figure 4.4-31. Data Flow Diagram

Building a system model by interviewing users usually begins with the processes defined at the primitive level and data defined in forms and manual files. Figure 4.4-32 illustrates part of a model built from user interviews. After building the model, the next task is to organize the data flows logically and then collapse the lower-level functions into higher-level functions. Figure 4.4-33 illustrates a logically organized version of the model built from interviews.

The DFD function titles, when wrapped in “shall statements,” become requirements statements.

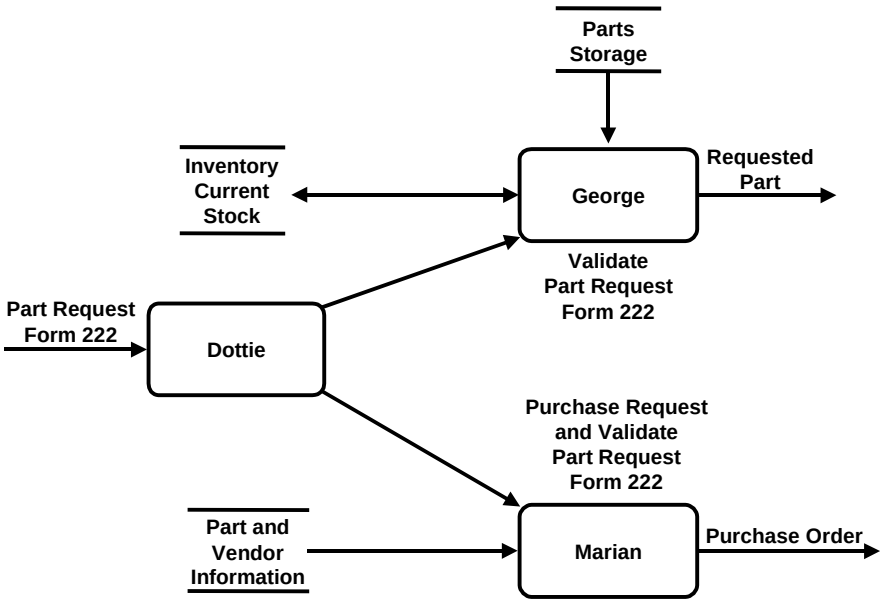


Figure 4.4-32. Primitive Data Flow from User Interviews

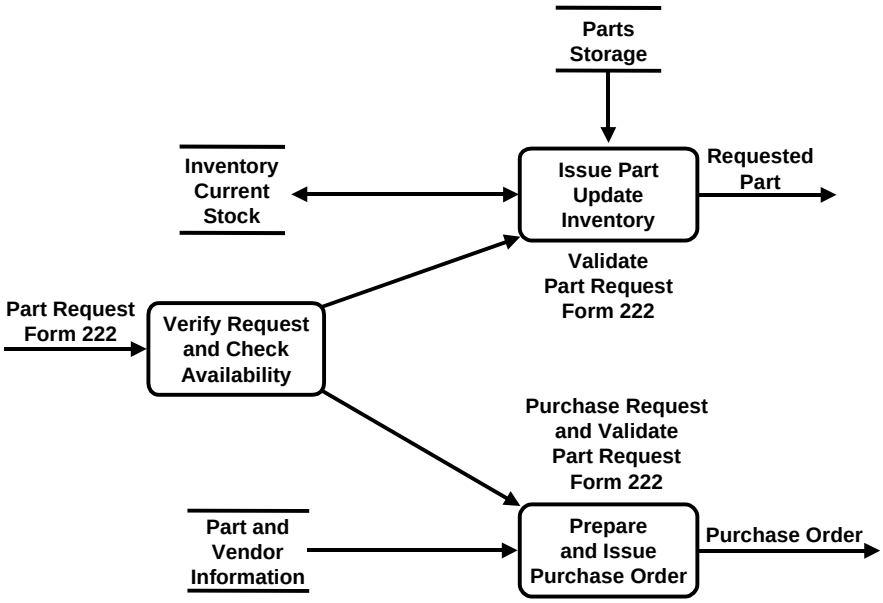


Figure 4.4-33. Logically Organized Data Flow

4.4.5.2.2.2. Data Dictionary

DDs are documents that provide a standard set of definitions of data flows, data elements, files, databases, and processes for a specific level of system decomposition. These documents aid communication across the development organization. The DD also defines data items

mentioned in the transformation specifications. For every data line and every data store illustrated on every DFD, a unique line in a DD that clearly defines the data item is required.

A DD defines the content of each data item, table, and file in the system. P-specs describe the capabilities that each process is required to provide. The specifications may be written in structured English and/or in the form of decision tables and decision trees. State diagrams graphically depict the logical states that the system may assume. Associated process descriptions specify the conditions that require fulfillment for the system to transition from one logical state to another.

When working from a set of customer documents, a top-down approach is used to decompose customer-defined processes. As each process is decomposed, so is the data. Only the data that a process requires to produce the specified outputs is documented in a DD. Functional decomposition usually proceeds to a level where the requirements for each lower-level function are stated on one page or less (i.e., the primitive level). Interaction with the customer may be necessary to decompose and define data elements at lower levels.

4.4.5.2.2.3. Process Specifications

For every lowest-tier bubble in the DFD analysis, it is necessary to write a p-spec that contains the p-spec for the bubble. This specification may be phrased in normal English text, structured English that follows a particular computer tool syntax, tables, state diagrams, or any combination of these constructs. P-specs, at the primitive level, when wrapped with “shalls” subsequently become requirements statements.

4.4.5.2.2.4. State Transition Diagram

After the DFDs and DD are complete, the next step is to identify the various states the system may assume and to produce diagrams depicting how the system transitions between states. It is suggested that a top-down approach, such as a state transition diagram (STD), be used to identify various states of the system, working down through the subsystem.

An STD is a graphical model of the dynamic behavior of a system—it is a sequential state machine that graphically models the time-dependent behavior of a control transformation. Figures 4.4-34 and 4.4-35 are examples of STDs for system and subsystem functions. Descriptions of how the system transitions from one state to another become “shall statements” in the requirements document.

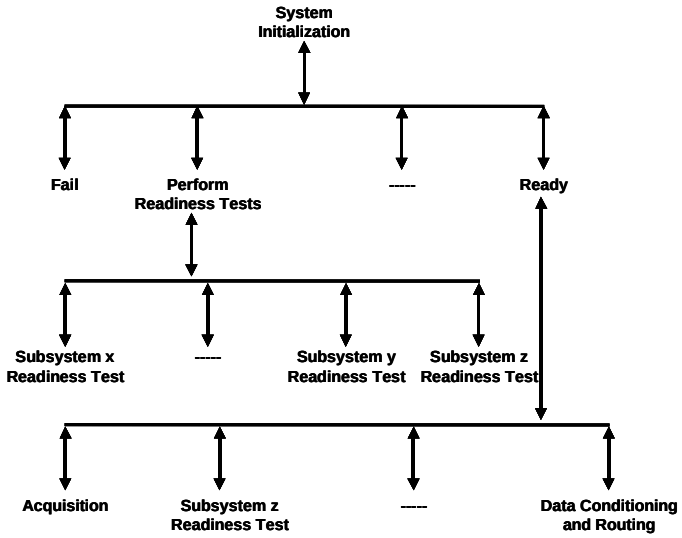


Figure 4.4-34. State Transition Diagram for System Functions

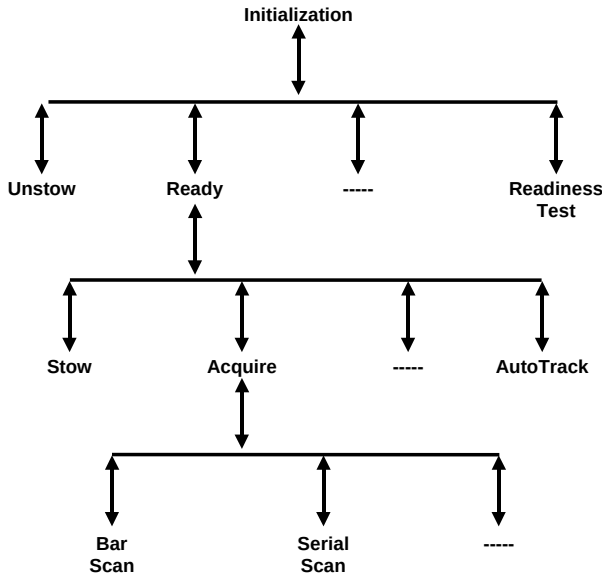


Figure 4.4-35. State Transition Diagram for Subsystem Functions

4.4.5.2.3. Hatley-Pirbhai Extension to Modern Structured Analysis

Because the traditional modern structured analysis process has proven inadequate for modeling real-time systems, the Hatley-Pirbhai Extension to modern structured analysis was created. This model extended the requirements model of modern structured analysis process to include an additional construct called a control flow diagram (CFD)—an augmentation of the corresponding DFD that has control as well as data processing functions. The CFD has the same bubbles as its companion DFD. The data lines that join the bubbles on the DFD are related only to the data associated with processing needs. The data lines shown on the CFD are only those data items related to control functions. This model may be considered a special

case of modern structured analysis and is particularly useful when the problem entails difficult control problems.

4.4.5.2.4. Models and Simulation

Models are abstractions of relevant characteristics of a system that are used to understand, communicate, design, and evaluate (including simulation) the system. They are used before the system is built and while it is being tested or in service. A good model has essential properties in common with the system/situations it represents. The nature of the properties it represents determines the uses for the model. A model may be functional, physical, and/or mathematical.

For complex system problems, it is necessary to analyze and design a number of different systems, each of which is represented by a specific model. The different models permit individual investigation of different aspects. These different modeling perspectives are incrementally constructed and integrated in a unified description (system model) to maintain a holistic system perspective from which the emergent properties of the system are deduced and verified.

The system model emphasizes the interactions of the objects in the context of the system, including the objects in the environment. Object semantics represent the components of a system, their interconnections, and their interactions when they are responding to the stimulus from the objects in the environment. These object semantics are partitioned into a static as well as dynamic modeling representation that describes the system's structure and behavior, respectively.

In this sense, the models embody the decisions made over the different steps of the Lifecycle Engineering process (Section 4.13). The models are developed as part of the decisionmaking process and support the evolution of the system design process as well as the iterative nature of the engineering in an environment where changes and enhancements to the models are managed in a controlled manner.

4.4.5.2.5. Thread Analysis

One major challenge to Functional Analysis is the development of software that implements the desired behavior of the system. Because system behavior is primarily implemented in software, a critical issue in system development is "how system engineers interact with the software engineers to ensure that the software requirements are necessary, sufficient, and understandable." This problem is addressed at the practitioner level, and experience has shown that the approach of passing paper specifications between systems and software developers does not yield satisfactory results.

Stimulus-condition-response threads are an excellent way to control the software development process, including translation from system to software requirements, design verification, review of software test plans, and integration of software and system testing. The threads enumerate the number of stimulus-condition-response capabilities to be tested. Threads also tie to performance requirements. Experience in the past 20 years on a variety of thread versions shows that such approaches are both feasible and effective.

4.4.5.2.5.1. The Use of Threads

System and software engineers shall work together to identify the system-level threads and the subset of the threads that the computer system supports. In this context, a thread consists of a system input, system output, description of the transformations to be performed, and conditions

under which the transformations hopefully occur. Such threads may be represented textually or graphically in a variety of ways, some of which are supported by tools. The following guidelines apply to the use of threads:

- The threads satisfy the need for efficient communication between system and software developers
- The identification of a thread from input to output allows the identification of the subthread to be allocated to the processing subsystem and, hence, software
- The description of stimulus-condition-response threads eliminates the ambiguities found in current specifications
- The description of threads is inherently understandable, particularly if provided in some graphical format
- The use of such threads aids in evaluating the impact of proposed changes

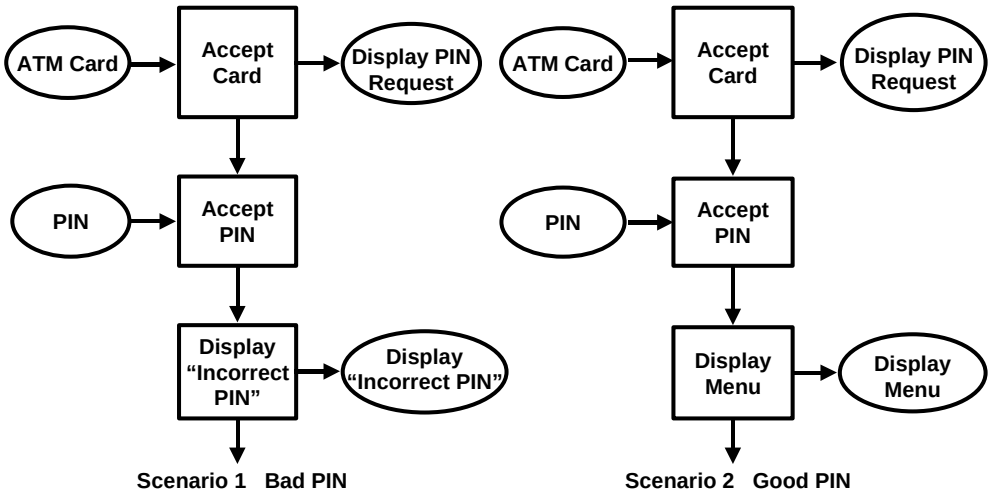
In the following steps, the development of software requirements is evolutionary, starting with allocation of processing requirements to a processing system and ending with publication and review of the software requirements.

4.4.5.2.5.2. Step 1: Deriving the System-Level Threads for Embedded Systems

No matter how the system description is developed, even if it is no more than the identification of system functions for different modes of operation, system inputs and outputs shall be identified in order to anchor the specification to reality. This process starts with the initial scenarios that describe the system's intended operations, which may be rewritten into the form of stimulus-condition-response threads.

To illustrate, consider the bank automated teller machine (ATM) system, which, by processing ATM cards and personal identification numbers (PIN), enables customers to perform banking transactions. Figure 4.4-36 presents two top-level scenarios that describe the top-level behavior of the ATM system when presented with an ATM card and a PIN. The two scenarios are PIN is accepted and PIN is rejected. From the scenarios or the integrated behavior, the stimulus-condition-response threads are identified. This set of threads may be specified in a number of notations. Figure 4.4-37 presents the stimulus-condition-response threads in a functional format. Note that the conditions for each of the threads are to avoid ambiguity. These conditions are a combination of two factors:

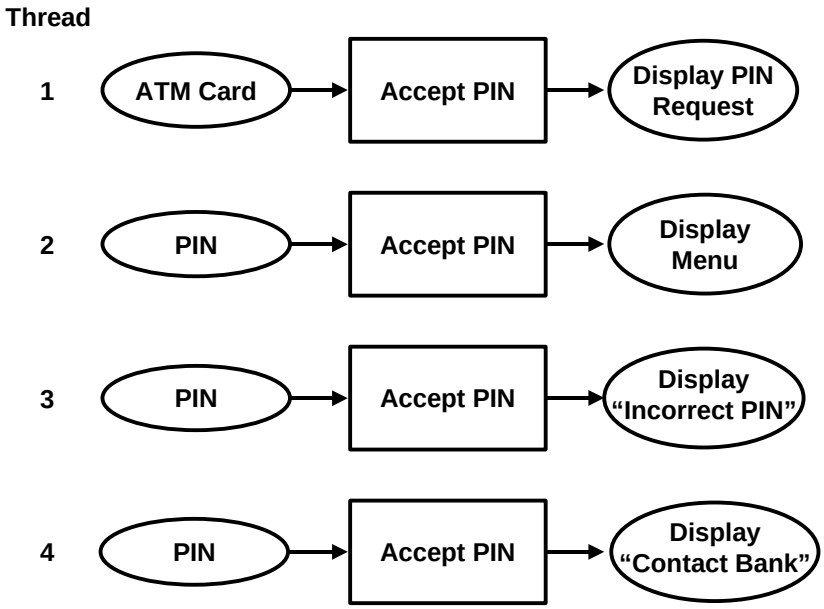
- The mode of the system, which determines which kind of input is expected
- The combination of values of the system state information and the contents of the input



Scenarios

Figure 4.4-36. Top-Level Scenarios in Thread Analysis

Thus, a correct PIN yields a menu, while an incorrect PIN results either in a message to "try again" or the machine "swallowing" the card, depending on the mode of the system. These conditions require that a thread be associated with the conditions in order to make them testable. To show the conditions explicitly, the "Accept PIN" function is decomposed to show its input-output behavior under different conditions.



Sample ATM Threads

Figure 4.4-37. Stimulus-Condition-Response Threads

4.4.5.2.5.3. Step 2: Decomposing the Threads to the Subsystem

It is necessary to decompose functions to the level where functions are uniquely identified by their requirements. This process is illustrated in Figures 4.4-36 through 4.4-40. In Figure 4.4-36, the top-level scenarios are defined; in Figure 4.4-37, examples of stimulus-condition-response threads are provided; in Figure 4.4-38, stimulus-condition-response threads are defined with their conditions; in Figure 4.4-39, the threads are defined in condition format; and in Figure 4.4-40, the system-level function "Accept PIN" is decomposed into functions to read the card (allocated to a card reader) and the functions and conditions allocated to the computer. Usually, most or all of the conditions are allocated to the computer system, with mechanical functions allocated to the other less intelligent components. Hence, most of the system threads yield a thread, with conditions, allocated to the computer subsystem, of which the majority is then allocated to the computer software with the software driving the computer hardware requirements. Thus, there is a direct traceable relationship between the system level, computer system level, and software level of requirements.

Figures 4.4-38 through 4.4-40 identify the difference between the system and computer system threads. The system uses a card reader component to read the card, a terminal component to accept push button inputs from customers, and a processor component to provide the intelligence to process the requests. Note that this process results in the requirement for the computer system to direct its threads to translate "card info" and "PIN info" to various output displays.

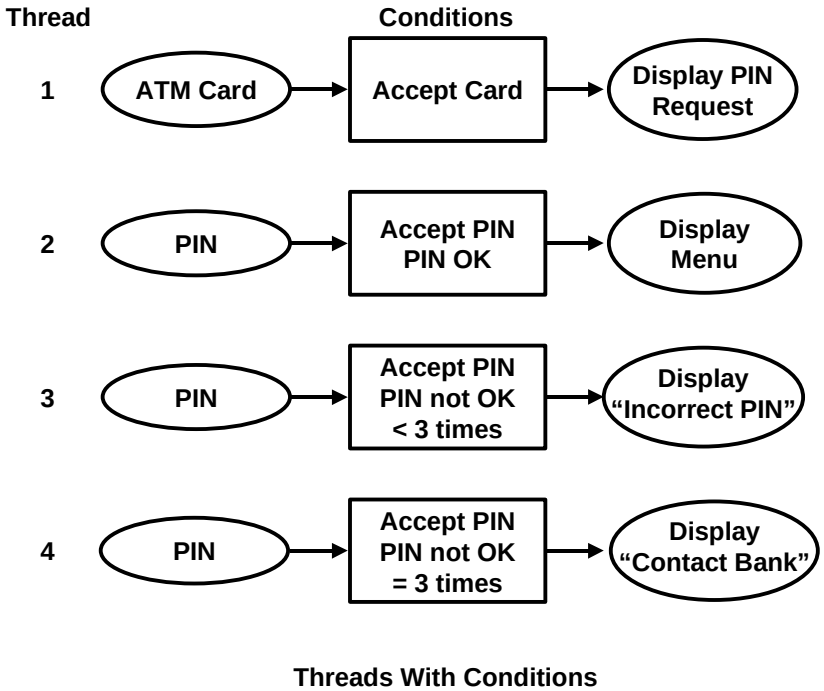
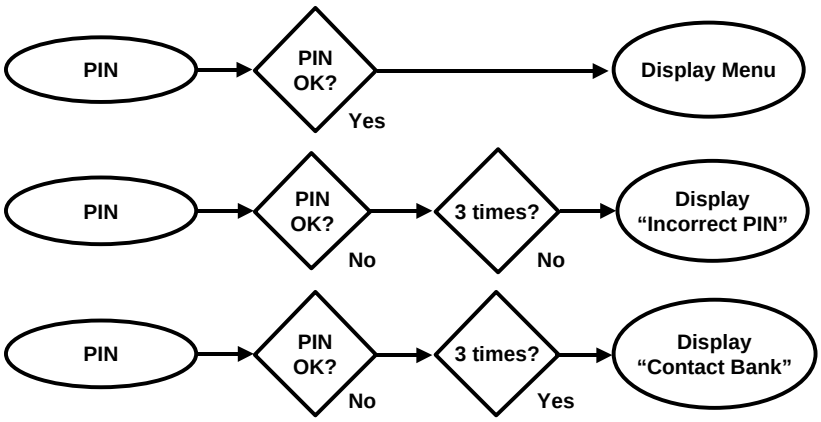
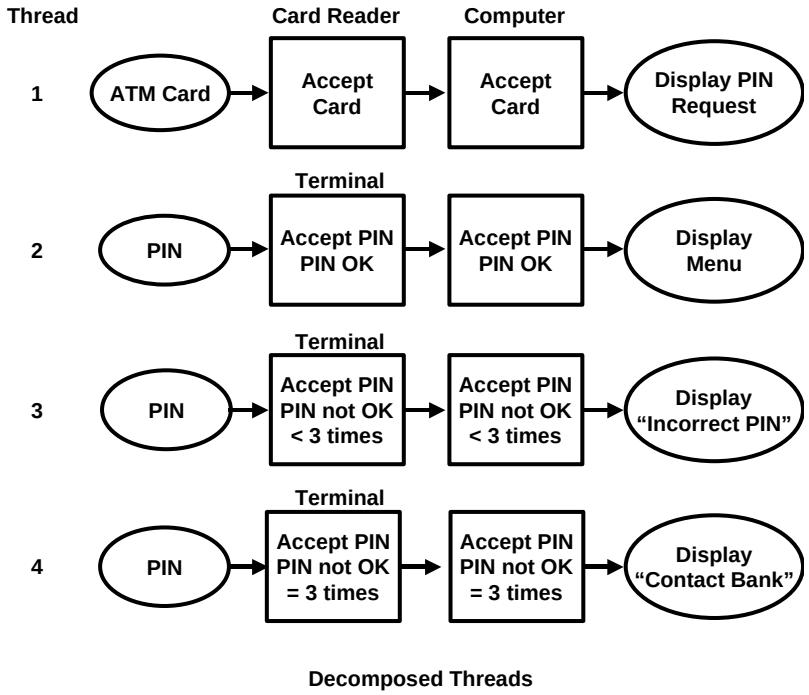


Figure 4.4-38. Threads Identified With Conditions



Threads in Condition Format

Figure 4.4-39. Threads in Condition Format



Decomposed Threads

Figure 4.4-40. System-Level Functions Decomposed

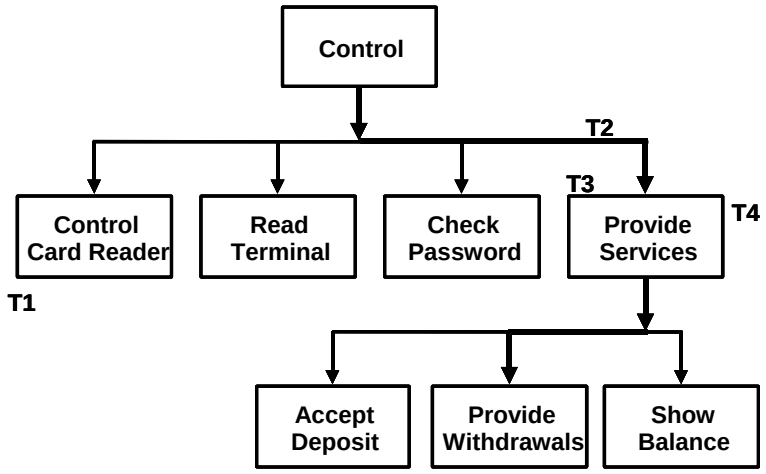
4.4.5.2.5.4. Step 3: Reviewing the Requirements and Design

Irrespective of the requirements format, computer system-level threads need to be traceable through the document. If a thread is untraceable, it signifies an omission in the requirements. If additional threads are identified that do not deal with interface designs or computer system-level fault detection/recovery, such threads may represent unnecessary processing; and, if so, it is

necessary to omit them. To illustrate, Figure 4.4-41 presents a simple software design in which a top-level program control calls lower-level units of code to execute operations.

Threads 1 through 4 are traceable through this design, thus validating it. The same approach works when an object-oriented design presents a number of objects implemented as independent software processes. When software design occurs that divides the overall software into computer software configuration items (CSCI), computer software components (CSC), and computer software units (CSU), the above process of decomposing and allocating the system-level threads into the components is repeated for each level of component. Again, if a thread is untraceable, it signifies an omission in the design. The system engineer traces the allocated system requirements to the software requirements review, which is followed by the CSCIs, CSCs, and CSUs for the software preliminary and critical design reviews.

If the software designers trace the computer system to software design threads as part of the requirements satisfaction demonstration, then the system engineer need only verify the completeness of the traceability. Tools strengthen the reliability of such a traceability evaluation. If the software designers do not perform this activity, then it is recommended that a joint team of system and software engineers perform the tracing to verify the design in preparation for the design reviews. In any event, the software-level threads shall be identified in order to provide systematic test planning.



Mapping S/W Threads onto a S/W Design

Figure 4.4-41. Tracing the Threads

4.4.5.2.5.5. Step 4: Tracing the Threads to the Test Plans

It is recommended that the collection of threads be exercised by the collection of tests outlined in the test plan—at the software level, computer system level, and the system level. This collection may be represented by a database and displayed in a cross-referenced matrix showing the relationship of system to software thread, software thread to software design threads, and threads to test cases at the various levels of integration, enabling the tools to ensure that every level of thread is tested at some point.

It is strongly recommended that the software threads be used to drive the test planning process using the concept of builds of software. For a system test, other components are added, and the system test threads are tested. For the ATM example, the difference is that in the software

only test, the software receives information in the format expected from the card reader; in the system test, the card reader component itself is used as the source of the data when an ATM card is the input.

This same approach is also used to construct the system-level test plans in a way that exploits the early availability of computer software that provides user-oriented capabilities. Thus, an early build of software may be integrated with a card reader to perform a test of Thread 1 through the system before the remainder of the software is developed. If the card reader is not available until later in the test cycle, then other threads may be tested first.

4.4.5.2.5.6. Notation

Several kinds of notations may be used for tracking the threads, but these notations usually fall under requirements- and design-oriented notations. Requirements-oriented notations describe the inputs, conditions, and outputs, while the design-oriented notations describe the threads with respect to the major design elements. Because both eventually describe the same stimulus-condition-response information, their use is essentially equivalent (though the design-oriented notation is more useful for actually defining the builds of software).

4.4.5.2.5.7. Conclusion

System engineers need to take the lead in constructing a sufficient process for system and software engineers to communicate, as it is the responsibility of the communicator to communicate in a language that the recipient understands. It is not feasible for system engineers to wait for software requirements methodologies to stabilize and accomplish this objective because software requirements and design techniques show no signs of stabilizing. The problem needs to be addressed within the existing context of multiple software requirements languages.

4.4.5.2.6. Object-Oriented Analysis

4.4.5.2.6.1. Early Versions

Early models advanced by Yourdon (2000) focused on objects that encapsulated computer processing and data, thus ending the separate analysis of these two previously inseparable facets of any computer software entity and, thereby, providing a tremendous improvement in software analysis. The model encouraged problem space entry using objects that represented the physical entities in the problem and solution space. Functionality and behavior of the problem space was explored based on these objects; therefore, it was not possible to follow the concept of form follows function when applying it. Early OOA models may be effective in analyzing systems with heavy precedence but problematic when exploring unprecedented problems.

Most authors who supported early OOA encouraged identification of objects that reflected elements of the problem space, that these objects be linked and organized into major subject areas, and that they be followed by refining the objects by identifying object functionality in terms of a DFD and behavior using state diagrams. Note that it was not easy to begin with functionality or behavior, rather one had to explore functionality and behavior in terms of previously defined objects.

4.4.5.2.6.2. Unified Modeling Language

4.4.5.2.6.2.1. Background

The UML is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems as well as for business process modeling.[1] The UML represents a collection of "best engineering practices" that have proven successful in modeling large and complex systems. Rational Software and its partners developed the UML, which is now an industry standard (www.omg.org/uml). It is widely supported, and there are numerous commercial packages available (www.incose.org) that may be used to develop UML-compliant models. These packages provide a collection of functionality ranging from purely drawing UML diagrams (low cost) to full round-trip engineering with model syntax checking and code generation (higher cost).

The principal benefit obtained from employing a standardized modeling language is that it provides a common framework for communicating system design and behavior between the organizations and various individuals, including users, architects/developers, and operators, involved with the system under development. Developing a model for an industrial-strength software system prior to its construction or renovation is as essential as having a blueprint for a building. Comprehensive models are essential for communication among project teams to ensure architectural soundness. As the complexity of the system increases, so does the importance of efficient modeling techniques.

The UML focuses on a standard modeling language, not a standard process. Although the UML has to be applied in the context of a process, experience has shown that various organizations and problem domains require a different process. The UML authors promote a development process that is use-case driven, functional architecture centric, iterative, and incremental. However, this specific development process is not required or enforced by the language. The UML merely provides the capability for:

- Model elements—fundamental modeling concepts and semantics
- Notation—visual rendering of model elements
- Guidelines—idioms of usage within the trade

Additionally, the UML provides extensibility and specialization mechanisms to extend core components. Though the UML is object-oriented by default, it is independent of particular programming languages.

4.4.5.2.6.2.2. Development Artifacts

The decision regarding which diagrams to create is largely dependent upon the system under development. Focusing on the relevant aspects of the system is critical in the abstraction process. The UML provides a rich notation to describe the static and dynamic behaviors of the system through several diagrams. These diagrams provide complementary views of the system, which are then developed and used by the various stakeholders.

The UML diagrams fall into the four following groups: use-case diagrams, class diagrams, behavior diagrams, and implementation diagrams.

---

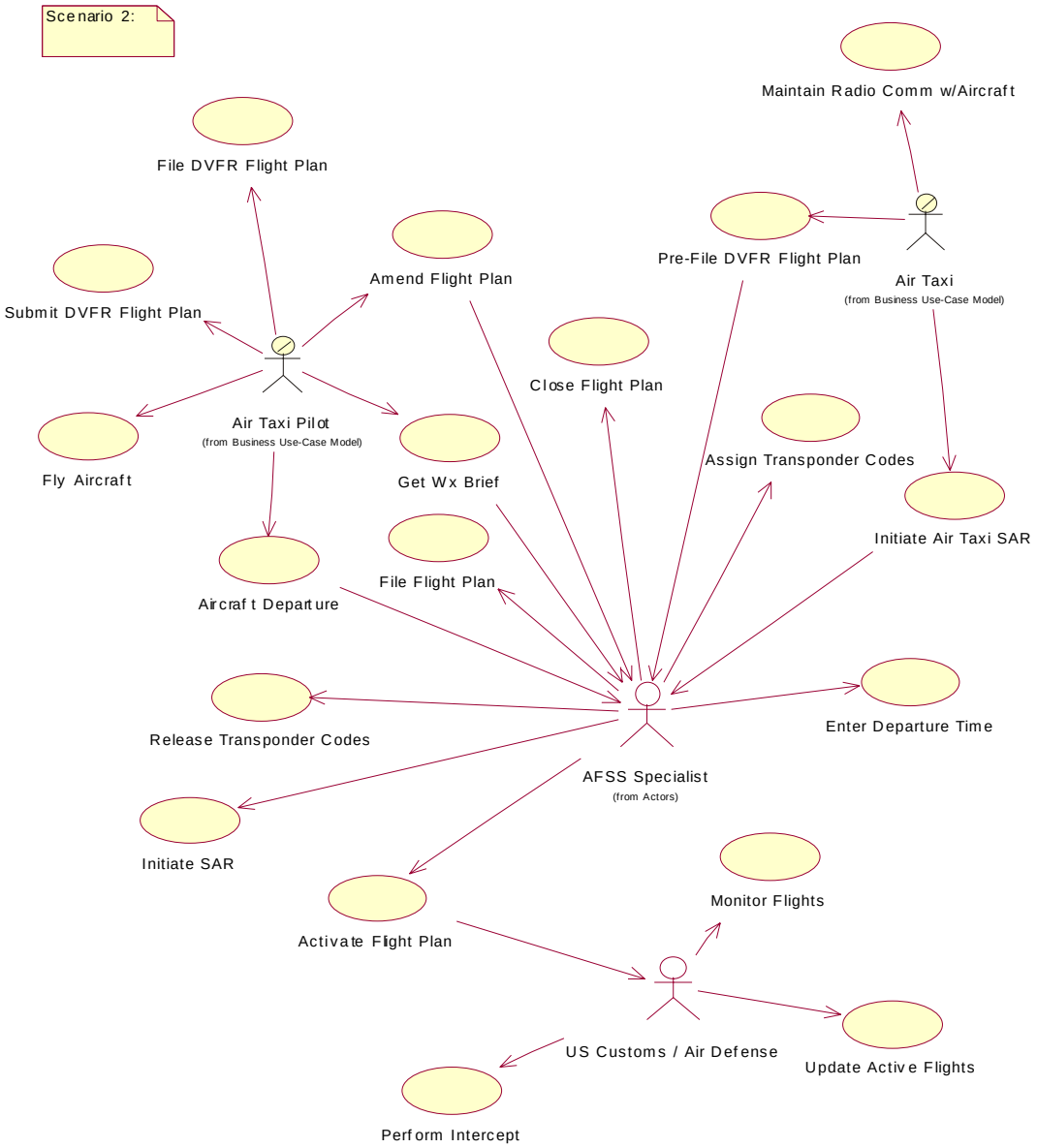
[1] OMG Unified Modeling Language Specification, Version 1.4, September 2001.

Use Case Diagram: A use-case diagram depicts one or more use-cases with its associated primary and secondary actors. An actor defines a role that a person plays with respect to the system. A use-case, by definition, yields an observable result of value to its primary actor. A secondary actor may be invoked by the use-case and provides a service. Actors may have a primary role in one use-case and a secondary role in another use-case. Use-cases are particularly well suited for capturing requirements. Figure 4.4-42 is an example of a use-case diagram.

Class Diagram: A class diagram provides a static view of the system's classes and depicts the relationships between the various classes. A class is a fundamental construct in all object-oriented languages and includes the notion of data and functions that are logically grouped. Individual class diagrams may depict attributes (e.g., data) and operations (e.g., functions) with varying levels of detail as necessary.

Behavior Diagrams: Behavior Diagrams are used to depict the dynamic operation of the system and include statechart diagrams, activity diagrams, sequence diagrams, and collaboration diagrams. A statechart diagram typically describes all possible states that a particular object may inhabit and how the object's state changes with regard to external events. Activity diagrams describe sequences of activities in which an activity typically represents a real-world process. Sequence diagrams depict a time-ordered flow of events between classes and actors and frequently describe a complex interaction between a small number of classes. Similarly, collaboration diagrams depict a time-ordered flow of events between actors and classes using a different layout; they are frequently drawn at a more abstract level. Collaboration diagrams are well suited for identifying underlying design patterns. Figures 4.4-43 and 4.4-44 are examples of behavior diagrams.

Implementation Diagrams: Implementation diagrams include both component and deployment diagrams. A component diagram depicts the various components and their dependencies in which a component typically represents a physical module of code. Alternatively, a deployment diagram depicts the physical relationships between software and hardware components.



1121

1122 **Figure 4.4-42. Day Visual Flight Rules Prefiled Flight Plan Use-Case Diagram**

1123

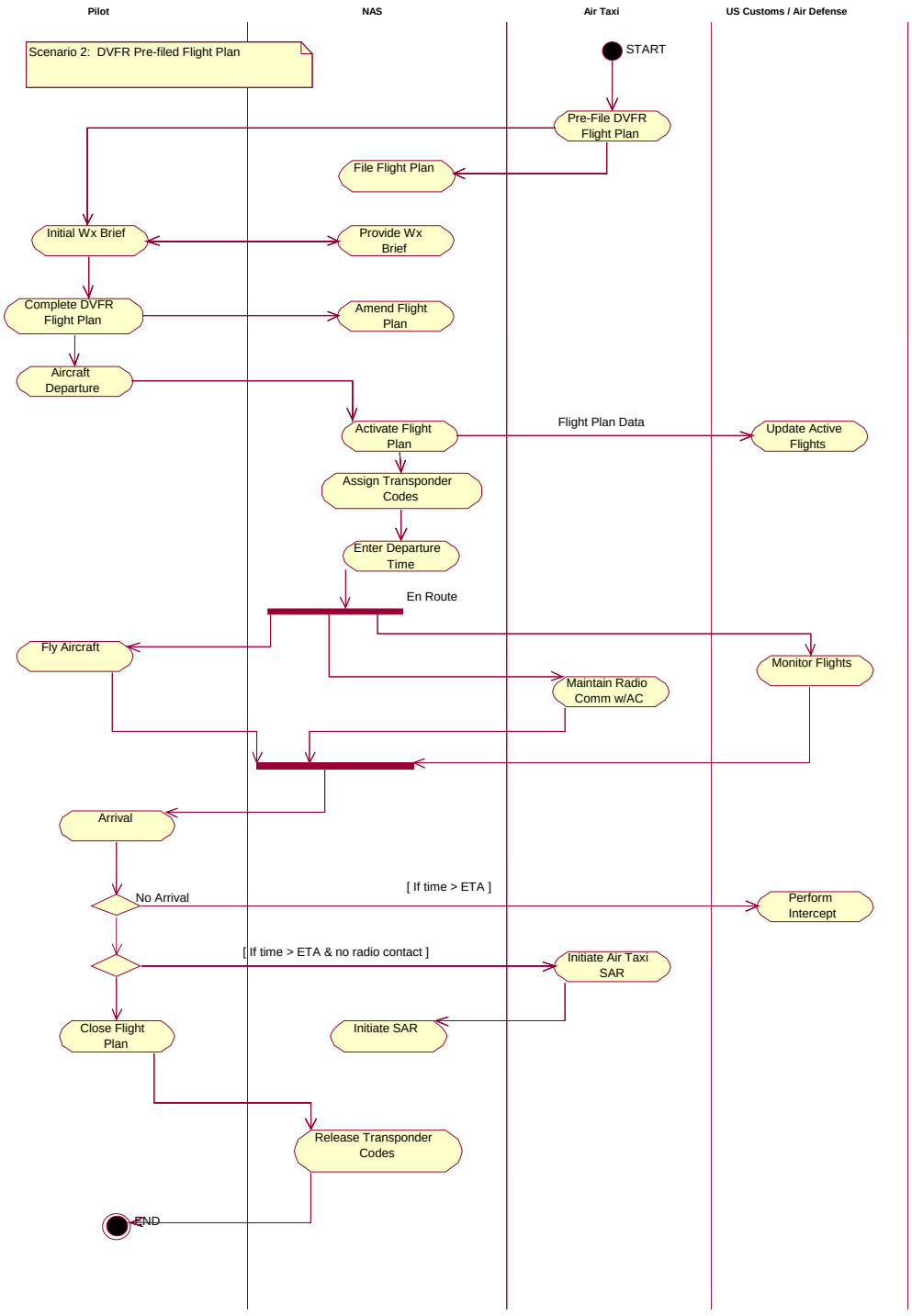


Figure 4.4-43. Day Visual Flight Rules Prefiled Flight Plan Activity Diagram

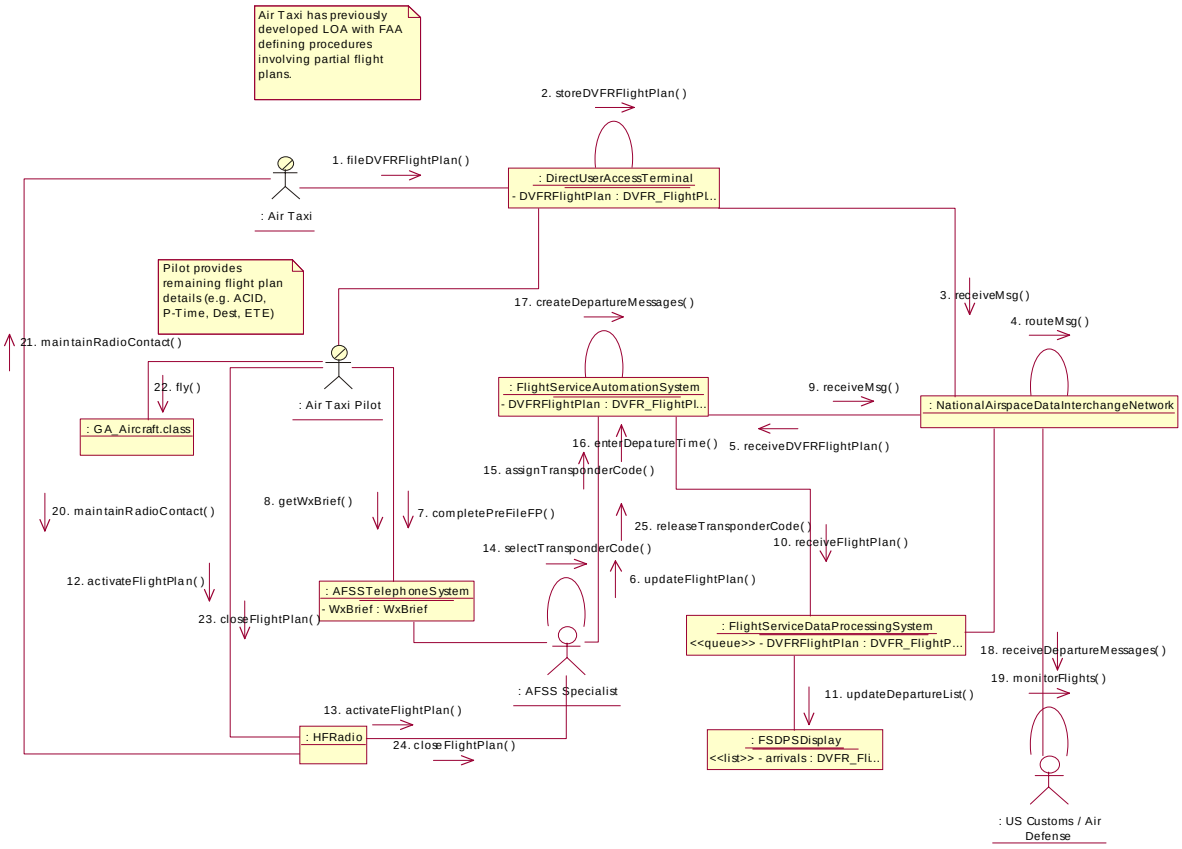


Figure 4.4-44. Day Visual Flight Rules Prefiled Flight Plan Collaboration Diagram

4.4.6. Functional Analysis Process Metrics

Candidate metrics used to measure the overall process and products of Functional Analysis include the following:

- Percent of analysis studies completed (schedule/progress)
- Depth of the functional hierarchy as a percentage versus the target depth
- Percent of performance requirements allocated at the lowest level of the functional hierarchy

Of the seven general measurement categories (see Table 4.1-1), the two that are applicable to Functional Analysis are Process Performance and Product Quality. In addition to the measures listed above, other candidate measures for Functional Analysis are provided in the table below. It is recommended that each effort tailor these measures and add other applicable project-specific measures to ensure the contribution of necessary information to the decisionmaking processes.

Table 4.1-1. Candidate Measures for Functional Analysis*

Schedule and Progress	Resources and Cost	Product Size and Stability	Product Quality	Process Performance	Technology Effectiveness	Customer Satisfaction
Achievement of specific milestone dates	Total effort compared to plan	Documentation of interfaces	Technical performance	Process productivity	Technology impact on product	Customer survey results
Test status	Resource utilization	Requirements	Defects	Process activity cycle time	Baseline changes	Performance rating
			Standards compliance	Defect containment		

*NOTE: These measures are only general examples to indicate the type of information that might be included in the individual section measurement matrix.

4.4.7. References

1. Blanchard, B. "System Engineering Management. 2nd ed. New York, New York: John Wiley & Sons, Inc.
2. Blanchard, B., and W. Fabrycky. "Systems Engineering and Analysis." 2nd ed. Englewood Cliffs, New Jersey: Prentice Hall.
3. Defense Systems Management College. "Systems Engineering Fundamentals." Fort Belvoir, VA: Defense Systems Management College Press, 1999.
4. Department of Defense. Military standard system safety program requirements. MIL-STD-882, 1984.
5. Federal Aviation Administration. "NAS Modernization System Safety Management Program." Washington, DC, 2001 < http://fast.faa.gov/toolsets/index2.htm>
6. International Council on Systems Engineering. "Systems Engineering Handbook," Version 2.0, 2000
7. National Aeronautics and Space Administration. "NASA System Engineering Handbook," June 1995.
8. RTCA, Inc. "Guidelines for the Approval of the Provisions and Use of Air Traffic Services Supported by Data Communications." RTCA DO-264.
9. SAE. "Certification Considerations for Highly-Integrated or Complex Aircraft Systems." Aerospace Recommended Practice ARP-4754, 1996.
10. SAE. "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment." Aerospace Recommended Practice ARP-4761, 1996.
11. Sage, Andrew B., and William B. Rouse, eds. "Handbook of Systems Engineering and Management." New York, New York: John Wiley & Sons, Inc., 1998.
12. Yourdon, Edward. "Modern Structured Analysis."